

Short Course: Solving PDE's on Overlapping Grids with the Overture Framework

Bill Henshaw

Centre for Applied Scientific Computing,
Lawrence Livermore National Laboratory,
Livermore, CA, USA  94551

www.llnl.gov/casc/Overture

Initially presented at DAMPT, University of Cambridge, December 2004.

Acknowledgments

Supported by

Department of Energy, Office of Science

MICS Program: Mathematical, Information, and Computational Sciences

Lawrence Livermore National Laboratory research and development funding

Current Overture developers

Kyle Chand

Bill Henshaw

**Overture and the CG PDE solvers are available for download
from**

www.llnl.gov/CASC/Overture.html

Documentation and installation instructions are also available at the above web site.

Short Course Summary

An Overview

- Main features of Overture
- Overlapping grids.
- A one-dimensional overlapping grid problem.
- Approximating derivatives on curvilinear grids.

Using the A++/P++ Array Class

- Array operations.
- Parallel array operations.

Overview of the Main Overture Classes

- The Overture graphics interface: windows, menus, dialogs and mouse buttons
- Mapping's
- Grid's and GridFunction's
- Operators

- Building component grids using the native geometry capabilities

CAD fixup and modification

- Fixing and modifying CAD files with **rap**

Component Grid Generation and CAD

- Component grid generation on CAD geometries
 - **mBuilder**: the mapping builder
 - **hype**: the hyperbolic grid generator

Overlapping Grid Generation

- **Ogen**: the overlapping grid generator

A Primer for the High-Level Interface to Grids, Grid-Functions and Operators

- MappedGrid examples
- Overlapping grid examples

Adaptive Mesh Refinement and Overlapping Grids

- Block structured mesh refinement

- AMR and overlapping grids
- AMR components of Overture
- AMR performance and examples

Solvers

Oges: Overlapping Grid Equation Solver

Ogmg: Multigrid solver for Overlapping Grids

- fast solution of elliptic boundary value problems

The CG (Composite-Grid) Suite of PDE solvers

- cgad : advection-diffusion solver.
- cgins : incompressible flow.
- cgcns : compressible flow with adaptive mesh refinement.
- cgmx : Maxwell's equations.
- cgmp : multi-domain multi-physics problems

Movies:

- model two-stroke engine
- shock hitting a collection of (rigid-body) cylinders (Euler with AMR).
- cylinders falling in a channel (INS with MG).

Overview: Overture is a collection of C++ classes that can be used to solve partial differential equations on structured, overlapping and hybrid grids.

Key features:

- provides a high level interface for rapid prototyping of PDE solvers.
- built upon optimized C and fortran kernels.
- provides a library of finite-difference operators: conservative and non-conservative, 2nd, 4th, 6th and 8th order accurate approximations.
- support for moving grids
- support for block structured adaptive mesh refinement
- extensive grid generation capabilities
- CAD fixup tools
- interactive graphics and data base support.
- PDE solvers built upon Overture include:
 - cgin : incompressible flow.
 - cgcns : compressible flow with adaptive mesh refinement.
 - ogen : overlapping grid generation.
 - cgm : Maxwell's equations.
 - cgmp : multi-physics problems

Overture

OverBlown
INS, CNS

Oges
Linear Solvers

Ogmg
Multigrid

Ogen
Overlapping

Ugen
Unstructured

AMR

Grids

GridFunctions

Operators

Mappings

CAD fixup
Grid Generation

rap, hype
mbuilder

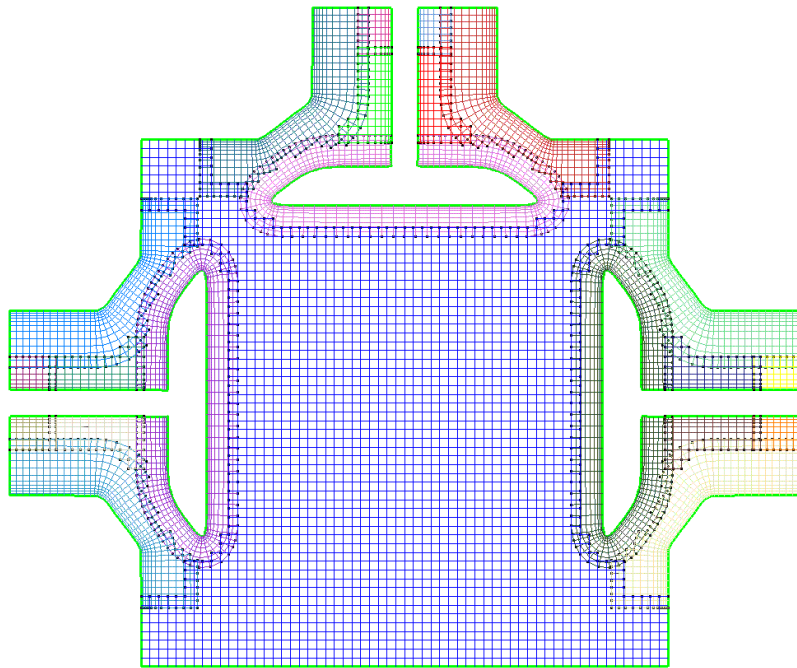
Graphics

A++/P++

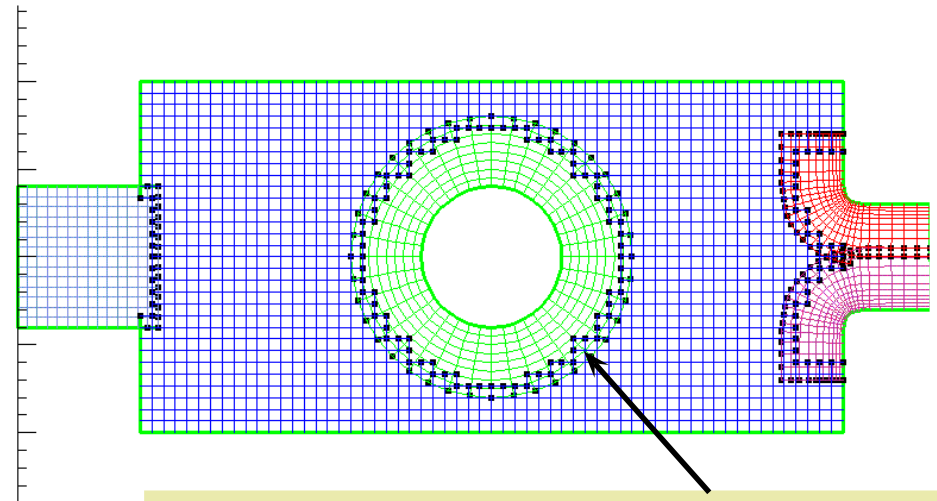
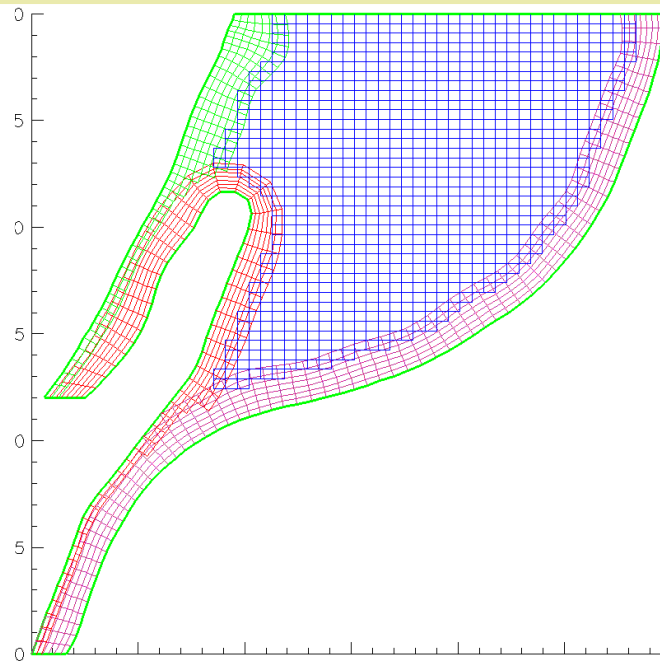
OpenGL
HDF

PETSc

Boxlib

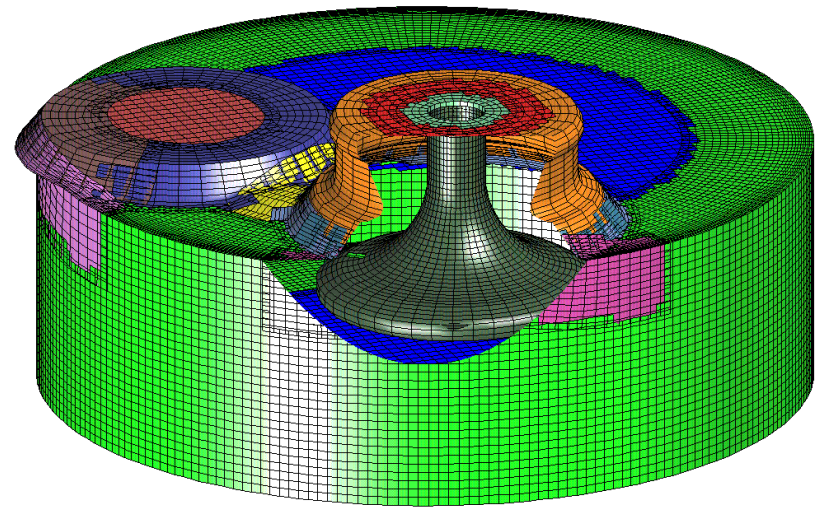
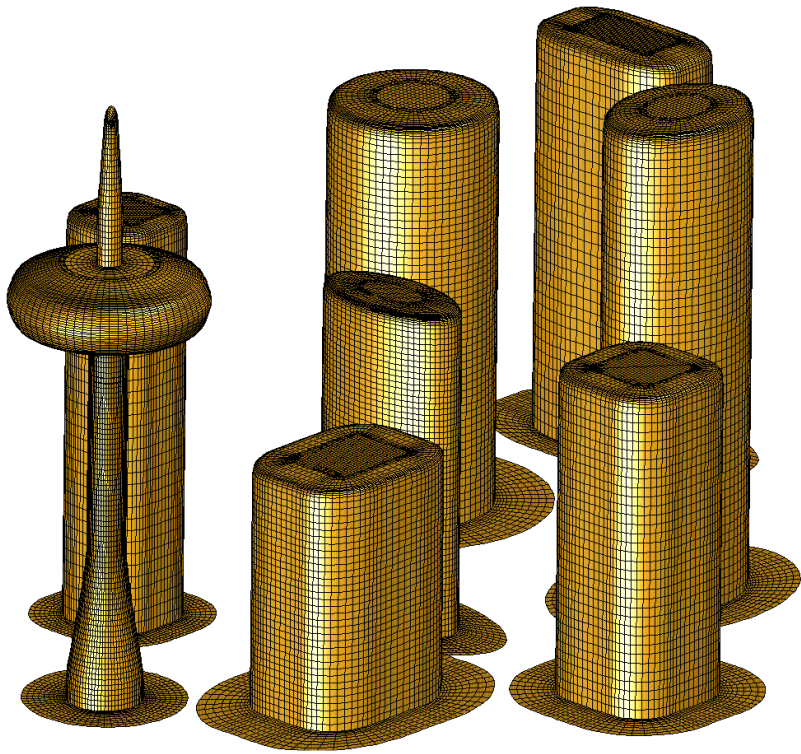


Sample 2D overlapping grids

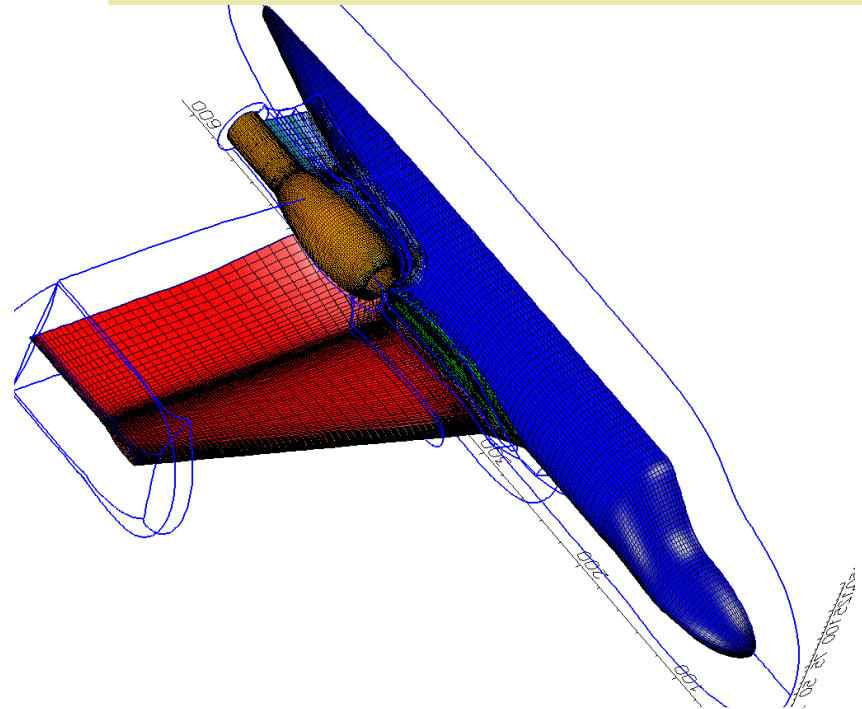
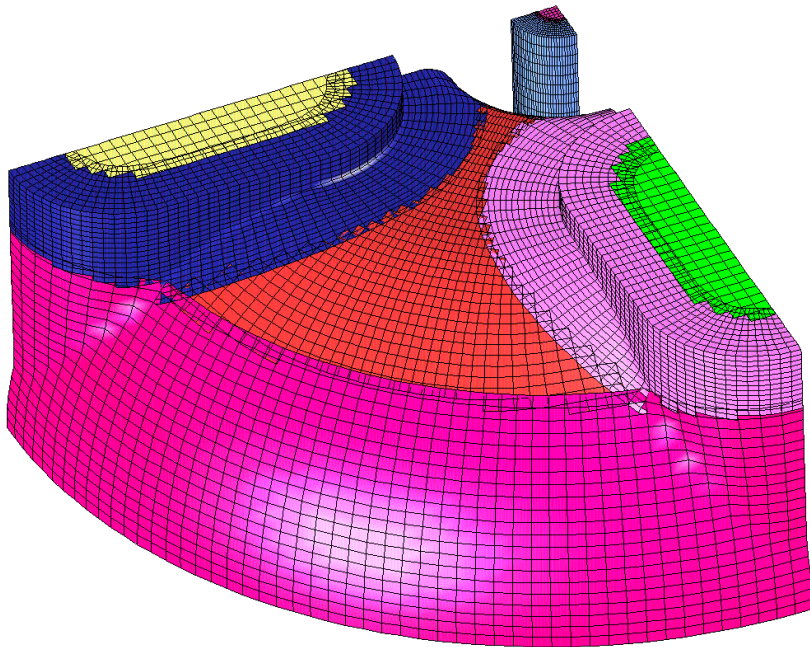


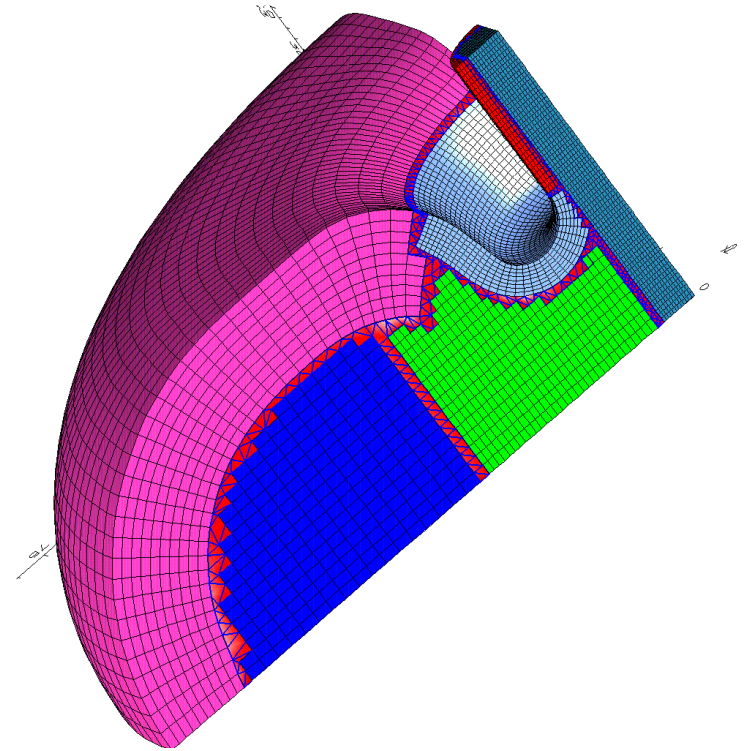
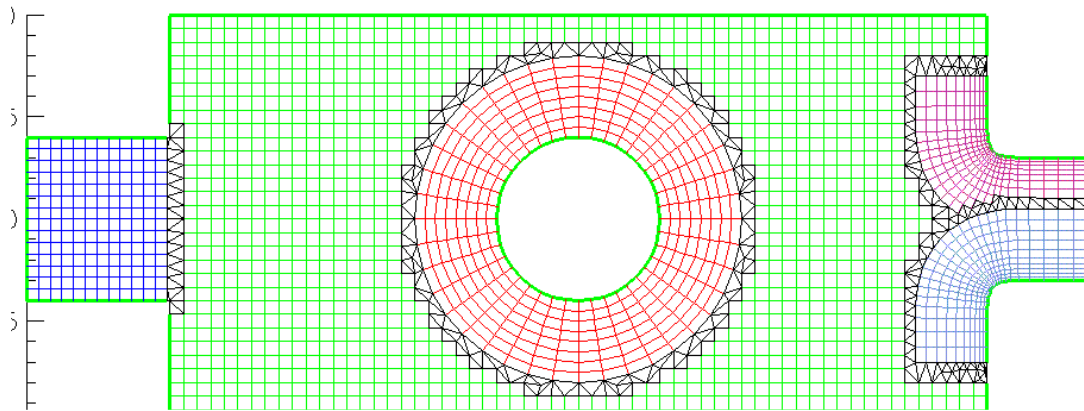
Solutions coupled by interpolation



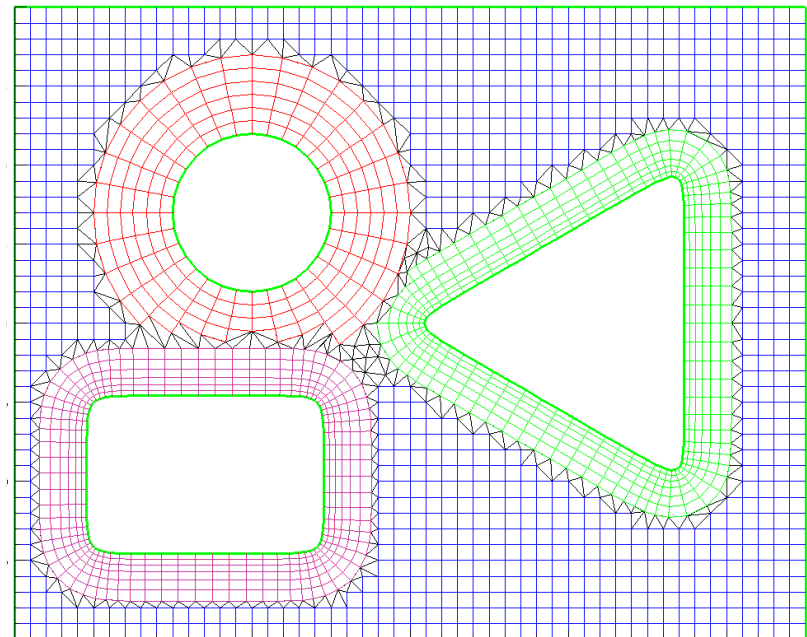
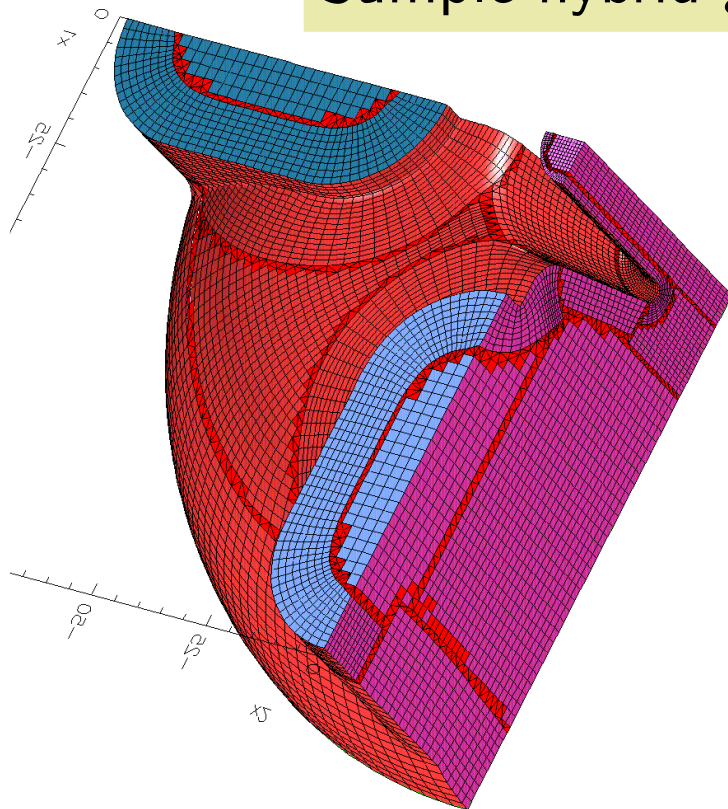


Sample 3D overlapping grids

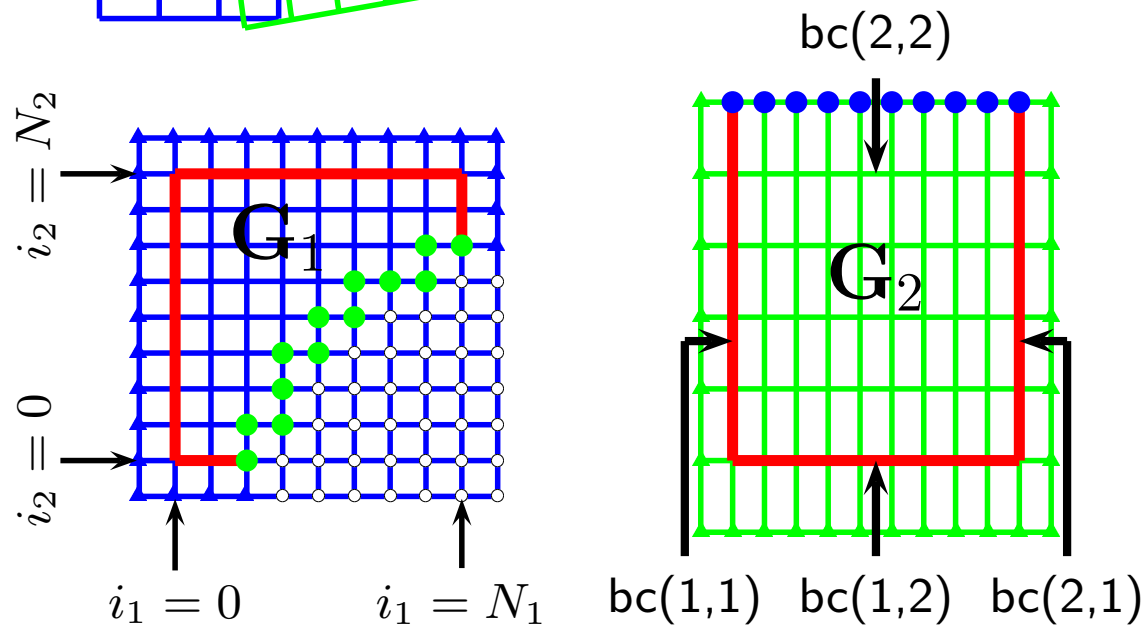
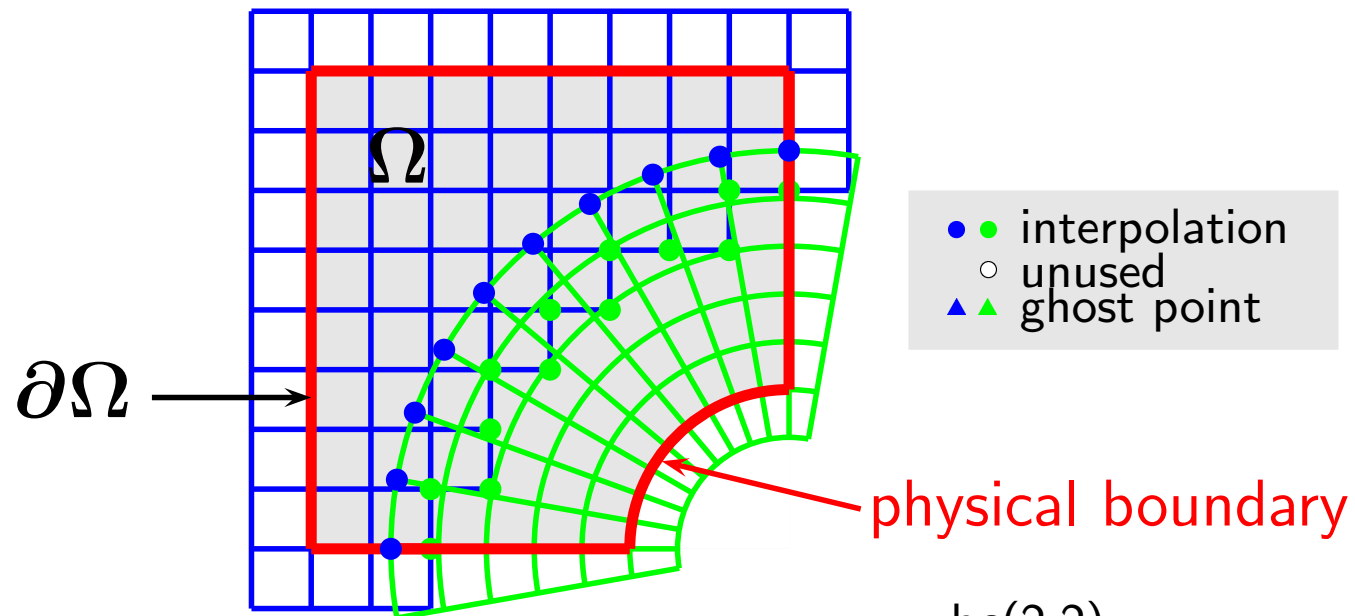




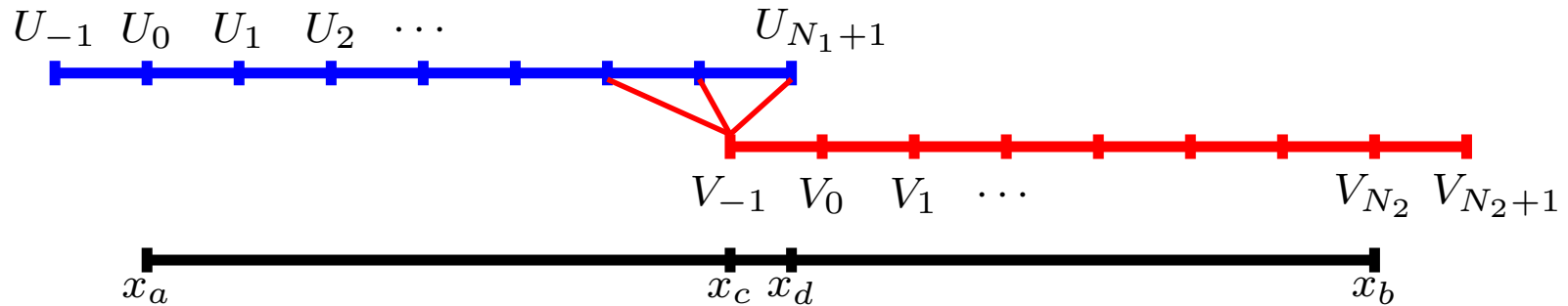
Sample hybrid grids



Components of an Overlapping Grid



A One-Dimensional Overlapping Grid Example:



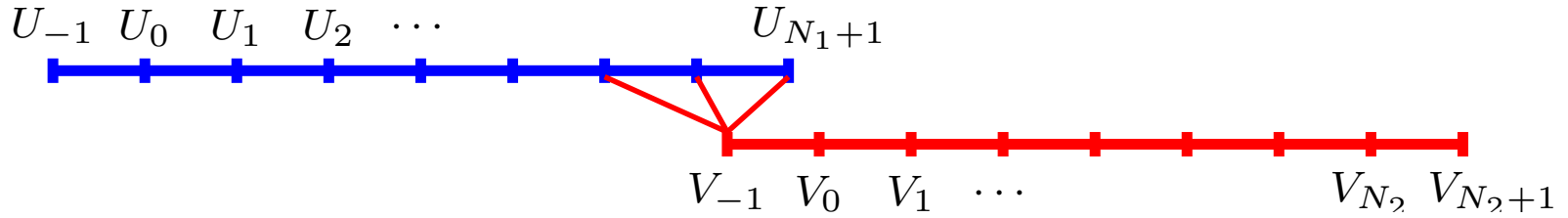
To solve the advection-diffusion equation

$$\begin{aligned}
 u_t + au_x &= \nu u_{xx} & x &\in (0, 1) \\
 u(0, t) &= g_0(t), \quad u_x(1, t) = g_1(t) & & \text{(boundary conditions)} \\
 u(x, 0) &= u_0(x) & & \text{(initial conditions)}
 \end{aligned}$$

introduce grid points on the two component grids,

$$\begin{aligned}
 x_i^{(1)} &= x_a + i\Delta x_1, & i &= -1, 0, 1, \dots, N_1 + 1, & \Delta x_1 &= (x_d - x_a)/N_1 \\
 x_j^{(2)} &= x_c + (j + 1)\Delta x_2, & j &= -1, 0, 1, \dots, N_2 + 1, & \Delta x_2 &= (x_b - x_c)/N_2
 \end{aligned}$$

and approximations $U_i^n \approx u(x_i^{(1)}, n\Delta t)$, $V_i^n \approx u(x_i^{(2)}, n\Delta t)$.



The equations discretized with forward-Euler time-stepping and central differences

$$(U_i^{n+1} - U_i^n)/\Delta t = -aD_0U_i^n + \nu D_+ D_- U_i^n \quad i = 1, 2, \dots, N_1$$

$$(V_j^{n+1} - V_j^n)/\Delta t = -aD_0V_j^n + \nu D_+ D_- V_j^n \quad j = 0, 2, \dots, N_2$$

$$U_0^n = g(t_n), \quad D_0V_{N_2} = g_1(t_n) \quad (\text{boundary conditions})$$

$$U_i^0 = u_0(x_i) \quad i = 0, 2, \dots, N_1 + 1 \quad (\text{initial conditions})$$

$$V_j^0 = u_0(x_j) \quad j = -1, 0, 2, \dots, N_2 + 1 \quad (\text{initial conditions})$$

$$U_{N_1+1}^{n+1} = \frac{1}{2}(1 - \alpha)(2 - \alpha) V_{-1}^{n+1} + \alpha(2 - \alpha) V_0^{n+1} + \frac{1}{2}\alpha(\alpha - 1) V_1^{n+1} \quad (\text{interpolation})$$

$$V_{-1}^{n+1} = \frac{1}{2}(1 - \beta)(2 - \beta) U_{N_1-1}^{n+1} + \beta(2 - \beta) U_{N_1}^{n+1} + \frac{1}{2}\beta(\beta - 1) V_{N_1+1}^{n+1} \quad (\text{interpolation})$$

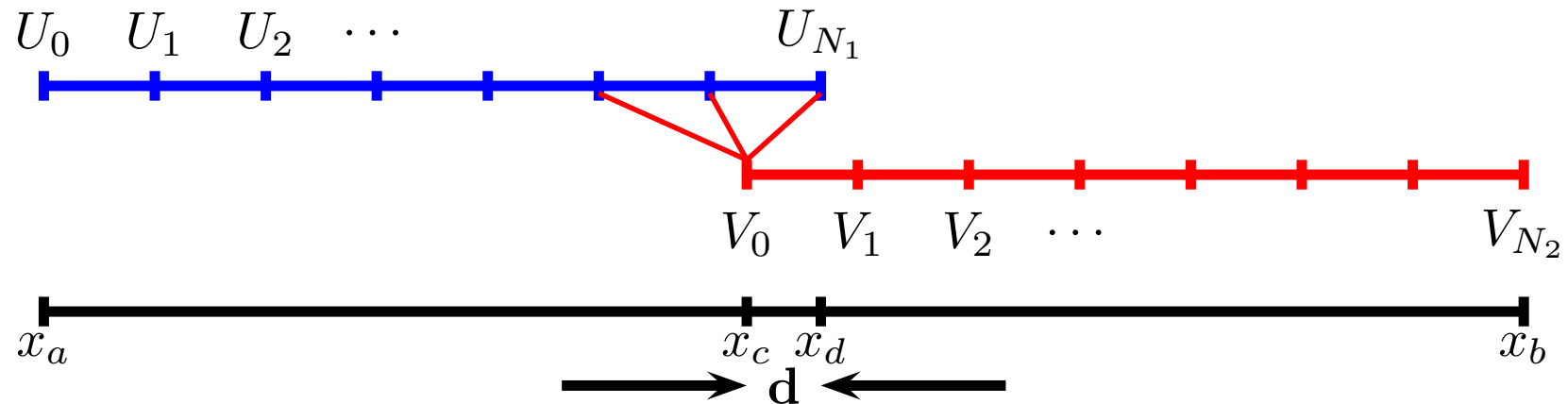
where $\alpha = (x_d - x_{-1}^{(2)})/\Delta x_2$, $\beta = (x_c - x_{N_1-2}^{(1)})/\Delta x_1$ and

$$D_0U_i^n = (U_{i+1}^n - U_{i-1}^n)/(2\Delta x_1),$$

$$D_+U_i^n = (U_{i+1}^n - U_i^n)/\Delta x_1,$$

$$D_-U_i^n = (U_i^n - U_{i-1}^n)/\Delta x_1$$

Interpolation between overlapping grids:



When solving Poisson's equation, $\Delta u = f$ with a scheme that is $O(h^{2p})$ the width of the interpolation formula should be

- width = $2p + 1$ if the overlap distance is $d = O(h)$.
- width = $2p$ if the overlap distance is $d = O(1)$.

Thus a second-order accurate scheme will normally require 3-point interpolation (quadratic interpolation).

Note: For a first order equation, $u_t + u_x = 0$, 2-point interpolation is sufficient for 2nd-order accuracy when the overlap distance is $O(h)$.

Overture supports a high-level C++ interface (but is built mainly upon Fortran kernels):

Solve $u_t + au_x + bu_y = \nu(u_{xx} + u_{yy})$

```
CompositeGrid cg; // create a composite grid
getFromADatabaseFile(cg,"myGrid.hdf");
floatCompositeGridFunction u(cg); // create a grid function
u=1.;
CompositeGridOperators op(cg); // operators
u.setOperators(op);
float t=0, dt=.005, a=1., b=1., nu=.1;
for( int step=0; step<100; step++ )
{
    u+=dt*( -a*u.x()-b*u.y()+nu*(u.xx()+u.yy()) ); // forward Euler
    t+=dt;
    u.interpolate();
    u.applyBoundaryCondition(0,dirichlet,allBoundaries,0.);
    u.finishBoundaryConditions();
}
```

Spatial approximations to derivatives:

Each grid is defined by a mapping $\mathbf{x} = \mathbf{G}(\mathbf{r})$ from the unit square $\mathbf{r} \in [0, 1]^d$ in d -dimensions to physical space $\mathbf{x} \in \mathbb{R}^d$.

Derivatives can be defined by the chain rule (with $\mathbf{r} = (r, s)$, $\mathbf{x} = (x, y)$). For example in two-dimensions,

$$u_x = r_x u_r + s_x u_s$$

$$\Delta u = (r_x^2 + r_y^2)u_{rr} + (s_x^2 + s_y^2)u_{ss} + 2(r_x s_x + r_y s_y)u_{rs} + (r_{xx} + r_{yy})u_r + (s_{xx} + s_{yy})u_s$$

Approximations to the derivatives using the *mapping-method* simply approximate the \mathbf{r} derivatives in the above expressions. For example, fourth order approximations are

$$u_r \approx D_{0r} \left(1 - \frac{1}{6} \Delta r^2 D_{+r} D_{-r} \right) u$$

$$u_{rr} \approx D_{0r} \left(1 - \frac{1}{12} \Delta r^2 D_{+r} D_{-r} \right) u$$

The *inverse* Jacobian derivatives r_x , r_y , s_x , s_y are given by the mapping. Higher derivatives such as $r_{xx} = (r_x)_x$, $s_{yy} = (s_y)_y$ are approximated in the same manner as for u .

Conservative or finite-volume type discretizations are based on the self-adjoint form

$$\nabla \cdot \mathbf{u} = \frac{1}{J} \left((J \nabla_{\mathbf{x}} r \cdot \mathbf{u})_r + (J \nabla_{\mathbf{x}} s \cdot \mathbf{u})_s \right)$$

$$J = \det(\partial \mathbf{x} / \partial \mathbf{r}) \quad (\text{Jacobian})$$

Advantages of Overlapping Grids

- structured grids permit accurate and efficient algorithms.
- boundary fitted grids allow high fidelity representations of the geometry.
- Cartesian component grids can be treated very efficiently.
- advantageous for moving geometry.
- grid generation is easier when grids are allowed to overlap, compared to generating multi-block structured grids.
- The ability to generate **smooth** grids for complex geometry is important to get good accuracy on many types of problems.

Some References: Theory of Numerical Methods

The stability and accuracy theory for finite difference approximations to initial-boundary-value problems for overlapping grids is primarily founded upon the well established theory for finite difference methods for a single grid. Stability theory is often divided into methods based on *energy estimates* and methods based on *mode analysis (GKS theory)*.

For a discussion of energy-estimates and GKS theory see, for example,

- Gustafsson, Kreiss and Sundström, *Stability Theory of Difference Approximations for Mixed Initial Boundary Value Problems. II*, 1972 [3].
- Gustafsson, Kreiss and Oliger, *Time Dependent Methods and Difference Methods*, 1995 [4].
- Strikwerda, *Finite Difference Schemes and Partial Differential Equations*, 1989 [15].

In many cases, the analysis of a nonlinear, variable-coefficient PDE in general geometry can be reduced to the consideration of a constant-coefficient half-plane problem.

For a discussion of well-posed problems and this reduction see, for example,

- Kreiss and Lorenz, *Initial-Boundary Value Problems and the Navier-Stokes Equations*, 1989 [12]

Some journal references related to Overture solvers:

For a general discussion of the overlapping grid approach as well as issues related to the order of accuracy of interpolation see

G. Chesshire and W.D. Henshaw, *Composite Overlapping Meshes for the Solution of Partial Differential Equations* 1990 [1].

A conservative interpolation method is developed in

G. Chesshire and W.D. Henshaw, *A Scheme for Conservative Interpolation on Overlapping Grids* 1994 [2]

Issues related to the determination of boundary conditions for high-order accurate schemes are considered in

W.D. Henshaw, H.-O. Kreiss and L.G.M. Reyna, *A Fourth-Order Accurate Difference Approximation for the Incompressible Navier-Stokes Equations*, 1994 [8].

W.D. Henshaw, *A Fourth-Order Accurate Method for the Incompressible Navier-Stokes Equations on Overlapping Grids*, 1994 [6].

Some journal references related to Overture solvers (cont'd):

The AMR scheme for overlapping grids with applications to high-speed reactive flow are covered in

W.D. Henshaw and D.W. Schwendeman, *An Adaptive Numerical Scheme for High-Speed Reactive Flow on Overlapping Grids*, 2003 [10]

The development of a multigrid solver with near *text-book* convergence rates is described in

W.D. Henshaw, *On Multigrid for Overlapping Grids*, 2004 [9].

A Short History of Composite/ Chimera/ Overset/ Overlapping Grids

- Volkov, circa [1966] developed a *Composite Mesh* method for Laplace's equation on regions with piece-wise smooth boundaries separated by corners. Polar grids are fitted around each corner to handle potential singularities.
- Starius, circa [1977] (student of H.-O. Kreiss) considered *Composite Mesh* methods for elliptic and hyperbolic problems – introduces a hyperbolic grid generator.
- Steger, circa [1980] independently conceives the idea of the overlapping grid, subsequently named the *Chimera* approach after the mythical Chimera beast having a human face, a lion's mane and legs, a goat's body, and dragon's tail. NASA groups develop grid generator PEGSUS, hyperbolic grid generation and flow solver Overflow (Steger, Benek, Suhs, Buning, Chan, Meakin, et. al.)
- [1980] B. Kreiss develops overlapping grid generator which subsequently leads to the CMPGRD grid generator [1983] (Chesshire, Henshaw) later leading to the Overture set of tools [1994].

A++/P++ Arrays...

A++/P++ : Multidimensional Arrays for C++

- Primary developer: D. Quinlan; help from K. Brislawn, B. Gunney and B. Miller.
- Fortran90/matlab style array operations.
- Serial and distributed parallel arrays.
- Code is compiled with A++ header files to run on serial machines.
- Code is compiled with P++ header files to run on parallel machines.
- Arrays can be passed to Fortran subroutines (Fortran storage order)

class Index: Index I(base,count,stride) : for indexing arrays

class Range Range R(base,bound) : for dimensioning (or indexing arrays)

class intArray, floatArray, doubleArray : multidimensional arrays (maximum 6 dimensions) distributed across selected processors (also referred to as **intDistributedArray, floatDistributedArray, doubleDistributedArray**).

intSerialArray, floatSerialArray, doubleSerialArray: serial arrays, duplicated across all processors

A++ : Code Example

```
Range R(0,9);    // define a range R=0..9
doubleArray a(R,R), b(10,10);    // declare and dimension arrays
Index I(1,8), J(1,8);    // define two Index's: I={1,2,...,8}, J={1,2,...,8}
b=1.;    // array assignment
// stencil operation:
a(I,J) = .25*( b(I+1,J) + b(I,J+1) + b(I-1,J) + b(I,J-1) );
// In the above statement b(I+1,J) is itself an array that is a "view" of b
// stencil operation by scalar indexing:
for( int j=J.getBase(); j<=J.getBound(); j++ )
    for( int i=I.getBase(); i<=I.getBound(); i++ )
        a(i,j) = .25*( b(i+1,j) + b(i,j+1) + b(i-1,j) + b(i,j-1) );
```

Using the where statement in A++

C++

```
realArray x(10,10), u(10,10);  
Range I(1,8), J(1,8);
```

...

```
where( x(I,J) >.5 )  
    u(I,J) = sin( 2.*Pi*x(I,J) );
```

Fortran

```
do j=1,8  
  do i=1,8  
    if( x(i,j) .gt. 0.5 )then  
      u(i,j) = sin( 2.*pi*x(i,j) )  
    end if  
  end do  
end do
```

Indirect addressing with A++

C++

```
RealArray u(10), v(3);  
intArray ia(3);  
ia(0)=2; ia(1)=4; ia(2)=7;  
Range l(0,2);  
...  
v(l)=u(ia)+2.*u(ia+1);
```

Fortran

```
do i=0,2  
  v(i)=u(ia(i))+2.*u(ia(i)+1);  
end do
```

Passing A++ arrays to Fortran

C++

```
extern "C" { void myfortran_( int & m, int & n, real & u); }
```

```
realArray u(20,10);
```

```
myfortran_( u.getLength(0), u.getLength(1), *u.getDataPointer() );
```

Fortran

```
subroutine myfortran( m,n,u )
```

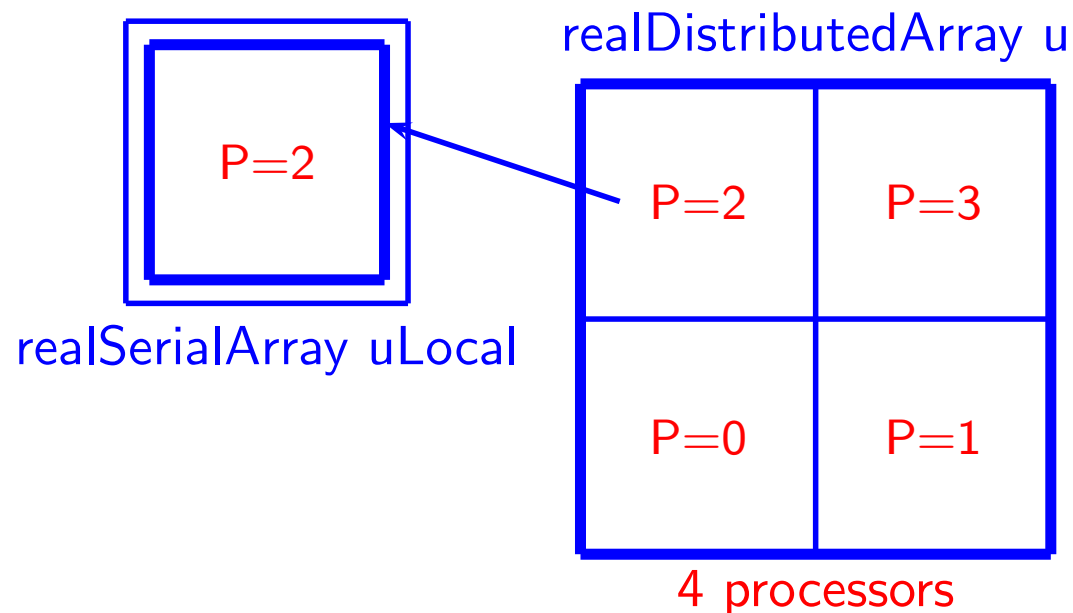
```
real u(m,n)
```

```
...
```

```
end
```

P++ : parallel multi-dimensional arrays

- ◇ Arrays are distributed across selected processors using a Partitioning object.
- ◇ Internal ghost-boundaries are added at the interfaces between processors.
- ◇ P++ uses Multiblock PARTI (A. Sussman, G. Agrawal, J. Saltz) for block structured communication with MPI (ghost boundary updates, copies between different distributed arrays)
- ◇ A serial array (local array) can be accessed from each processor.
- ◇ Caveat: Currently best results are obtained by operating on the local arrays and explicitly calling *updateGhostBoundaries* to update the ghost boundaries.



P++ : Sample Code

```
Range P(1,5); // Range of processors to use
```

```
Partitioning_Type partition(P); // object that defines the parallel distribution  
partition.SpecifyInternalGhostBoundaryWidths(1,1);
```

```
realDistributedArray u(100,100,partition); // build a distributed array  
u=5.;
```

```
realSerialArray & uLocal = u.getLocalArray(); // access the local array  
uLocal = cos(uLocal)+5.; // operate on local array
```

```
// call fortran with local array:
```

```
myFortranRoutine(*uLocal.getDataPointer(),...);
```

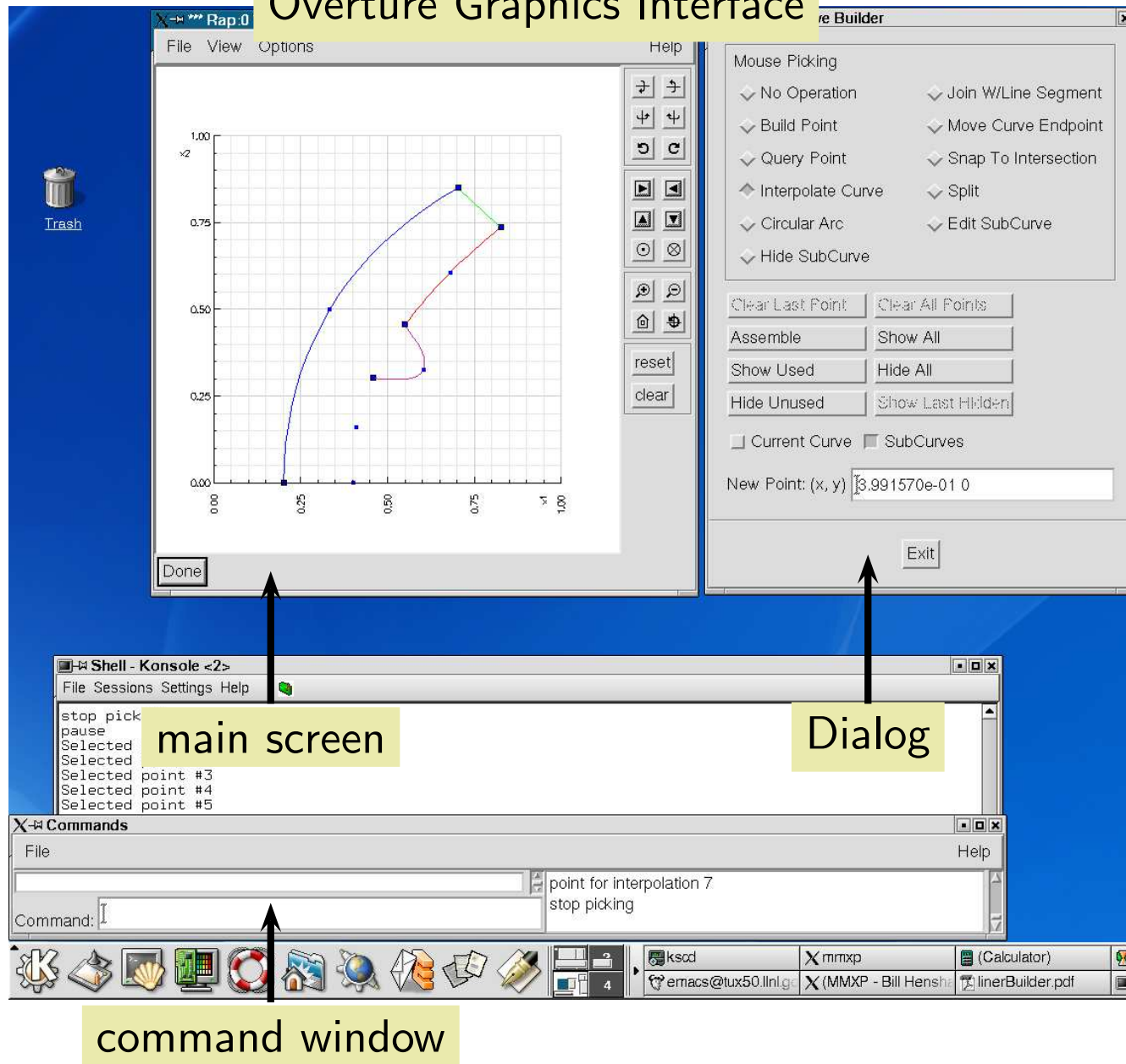
```
u.updateGhostBoundaries(); // update ghost boundaries between processors
```

**Graphics, Mappings, Grids, GridFunctions,
Operators, CAD...**











The Overture Graphics Interface

- ◇ Built upon OpenGL and Motif
- ◇ Mouse driven rotation, zoom, and picking (selection).
- ◇ High-level routines for plotting grids, contours, stream-lines.
- ◇ Interactive plotting by C++ calls.
- ◇ User defined menus and dialogs.
- ◇ Program control remains generally with the application, the GUI *event loop* is entered only when input is desired.
- ◇ OpenGL calls are restricted to the Graphics Interface classes. Motif is not exposed, usage is restricted to a few functions.

Overture Graphics Interface



Features of the graphical user interface

- User defined dialog windows that can contain pulldown menus with push or toggle buttons, option menus, text labels (for inputting strings), push buttons, and toggle buttons.
- Multiple graphics windows and a single command window with a scrollable sub-window for outputting text, a command line, and a scrollable list of previous commands.
- Rotation buttons  ,  ,  ,... which rotate the object on the screen about fixed x, y, and z axes (the x axis is to the right, the y-axis is up and the z-axis is out of the screen).
- Translation buttons  ,  ,  ,... which shift the object on the screen along a given axis.
- Push buttons for making the objects bigger:  , or smaller:  , and a reset button:  to reset the view point, and a **clear** button to erase all objects on the screen.
- A push button  to set the rotation center.
- A **rubber band zoom** feature.
- Mouse driven **translate, rotate and zoom**.
- A pop-up menu that is active on the command and graphics windows. This menu is defined by the application (= user program).

- Static pull-down menus (**file**, **view**, and **help**) on the graphics windows. Here, the screen can be saved in different formats, clipping planes and viewing characteristics can be set, annotations can be made (not fully implemented), and some help can be found.
- Static pull-down menus (**file** and **help**) on the command window. Here command files can be read/saved, new graphics windows can be opened, the window focus can be set, and the application can be aborted.
- A file-selection dialog box
- The option of typing any command on the command line or reading any command from a command file. All commands can be entered in this fashion, including any pop-up or pull-down menu item or any of the buttons, **x+r**, **y-r**, **x+**, **y+**, **bigger**, etc.
- Recording or retrieving a command sequence in a command file.

Mouse buttons and selection

The mouse driven features are summarized in table 1.

Modifier	Mouse button	Function
	left	picking
	middle	rubber band zoom
	right	pop-up menu
<SHIFT>	left	translate
<SHIFT>	middle	rotate around the x & y axes
<SHIFT>	right	zoom (up & down) and z-rotation (left & right)

Table 1: Mouse driven features.

Geometry, Grids and GridFunctions: Overview

Mapping : used to define continuous transformations. Example: the transformation from the unit square to an annulus.

MappedGrid: defines a grid for a Mapping; contains the grid points, Jacobian derivatives etc.

{int,float,double}MappedGridFunction : holds field values on a MappedGrid. Example: the density or temperature at each point on the grid. Derived from an A++/P++ array.

GridCollection : a list of MappedGrid's. Example: the grids forming an adaptive mesh refinement level.

{int,float,double}GridCollectionFunction : a list of MappedGridFunction's.

CompositeGrid : a GridCollection with interpolation information that represents an overlapping grid.

{int,float,double}CompositeGridFunction: holds field values on a CompositeGrid.

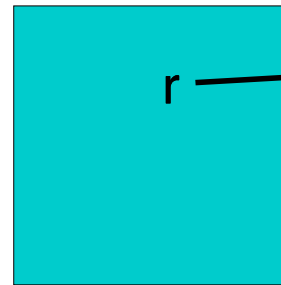
A Mapping defines a continuous transformation

Each mapping has an optimized **map** and **inverseMap**

Mapping

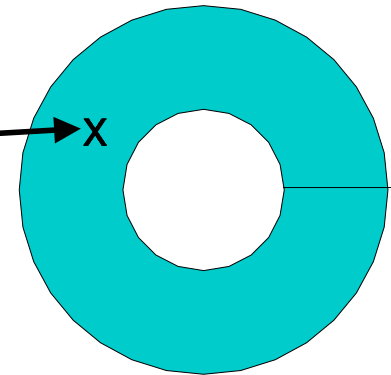


domainDimension
rangeDimension
map(r,x,xr)
inverseMap(x,r,rx)
boundaryConditions
singularities
...



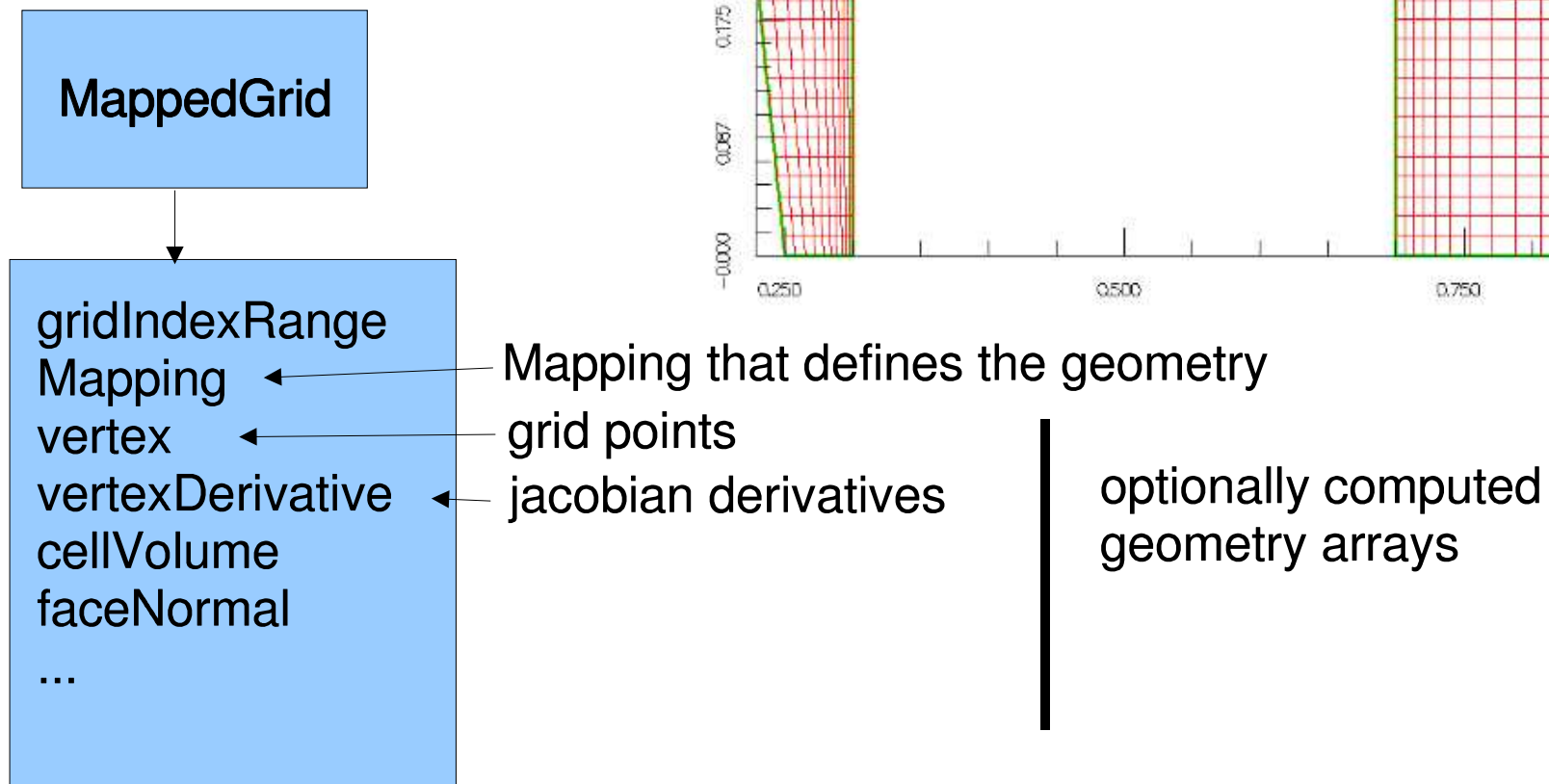
unit square

map

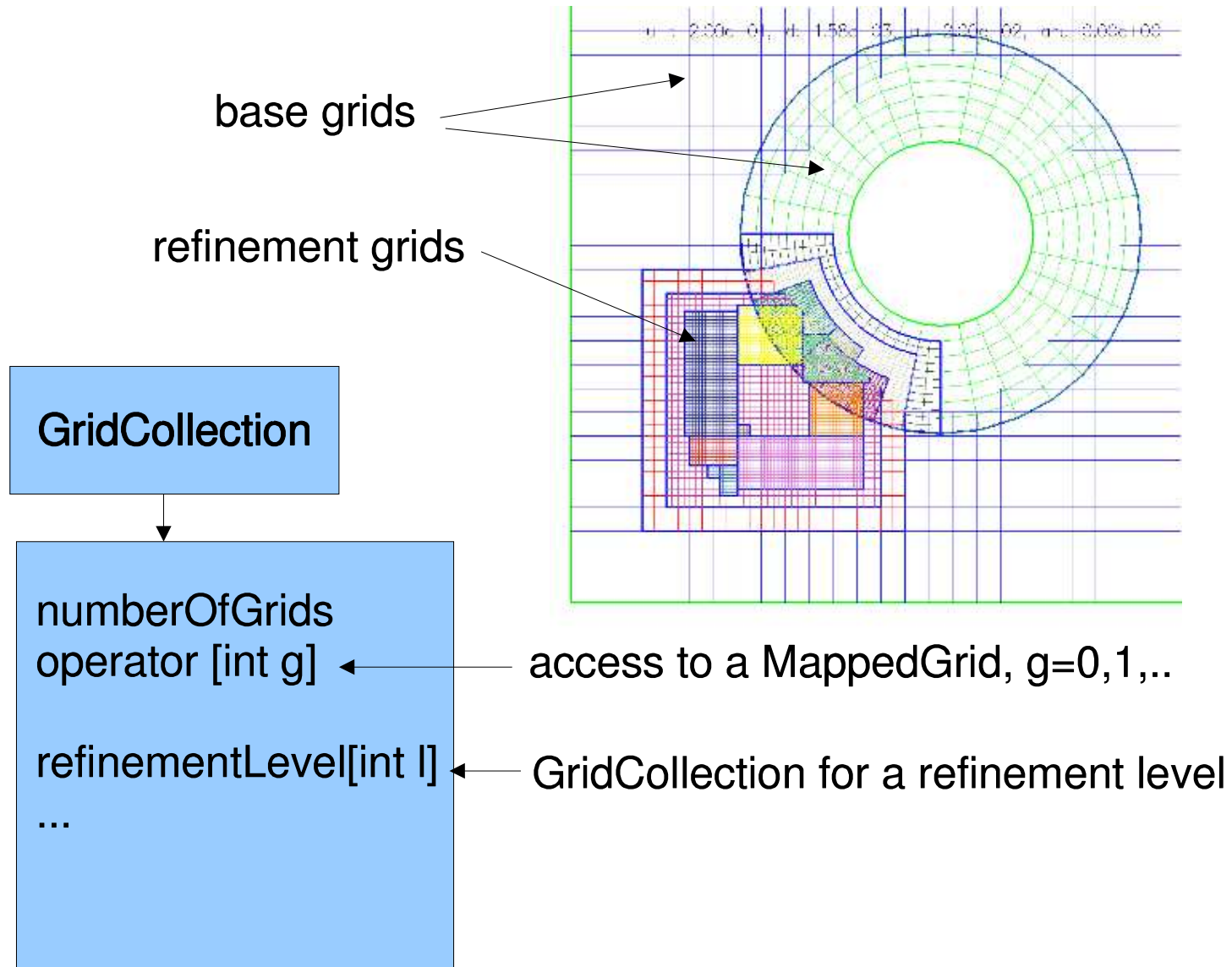


SquareMapping
AnnulusMapping
SphereMapping
HyperbolicMapping
EllipticTransform
MatrixMapping
RevolutionMapping
etc.

A MappedGrid holds the grid points and other geometry info



A GridCollection holds a list of MappedGrids



A MappedGridFunction holds solution values

realMappedGridFunction

derived from an A++ realArray
which implies it inherits all A++ operators

numberOfComponents ←

scalar, vector, 2-tensor, ..

MappedGrid

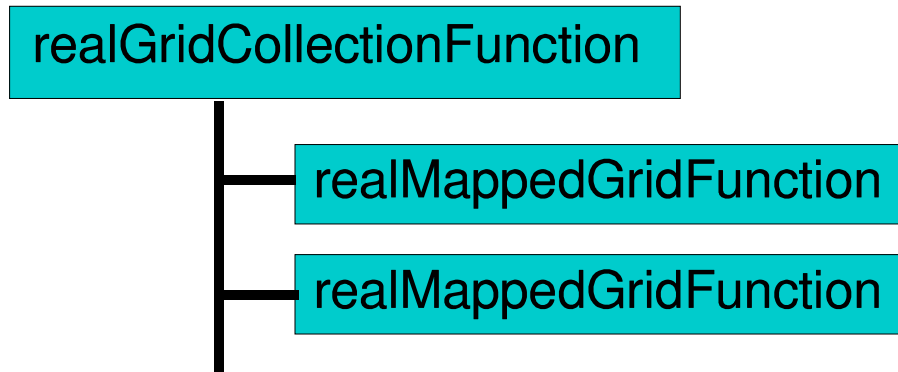
MappedGridOperators

x,y,z,xx,... derivatives

Grid functions can be vertex-centred,
cell-centred, face-centred etc.

```
MappedGrid mg(mapping);  
realMappedGridFunction u(mg);  
u=1.;
```

A GridCollectionFunction is a list of MappedGridFunctions



```
GridCollection gc(...);  
Range all;  
  
realGridCollectionFunction u(gc,all,all,all,2);  
  
u=1.;  
  
for( int grid=0; grid<gc.numberOfWorks(); grid++ )  
    u[grid]=1.;
```

Overview of the Operator Classes in Overture

Operators define discrete approximations to derivatives and boundary conditions.

- approximations for $\partial_x, \partial_y, \partial_z, \partial_{xx}, \dots, \Delta, \nabla \cdot (a \nabla)$, (soon higher derivatives too)
- orders of accuracy 2,4,6,8
- finite difference and finite volume approximations
- operators can be applied explicitly on a grid function or the sparse matrix representation of the operator can be formed.

Operators can be used at different levels

- CompositeGridFunctions: $u.x()$ – all points on all grids
- MappedGridFunctions: $u.y()$ – all points on one grid
- function call: `derivative(u,xOperator,...)` (avoids creation of some temporary arrays)
- macros or statement functions for evaluation point by point in C or Fortran codes

Operators

The Operator classes define differential operators and boundary conditions.

- 2nd/4th order approximations plus some conservative approximations

```
MappedGrid mg(sphere);
```

```
MappedGridOperators op(mg);
```

```
floatMappedGridFunction u(mg), v(mg), w;
```

```
v=u.x()+u.y();
```

← Compute the derivatives directly

```
w=u.laplacianCoefficients();
```

← Form the sparse matrix for the differential operator

Overview of the Geometry Capabilities in Overture

Native Geometry capabilities

- Curves, surfaces and volumes are represented with the **Mapping** class
- Many analytic representations including rectangle, annulus, cylinder and sphere
- Spline and NURBS representations are available.
- Transfinite interpolation can be used to interpolate curves or surfaces to form 2D-regions or 3D volumes.
- Transformations such as rotation, scaling, translation, body-of-revolution, stretching-grid-lines, sweeping, and extruding are also represented as a **Mapping**.
- Mappings can be composed: e.g. compose a annulus-mapping with a stretch-mapping to get a new mapping for an annulus with clustered grid-lines.
- 3D surfaces can be intersected to form one or more curves of intersection.

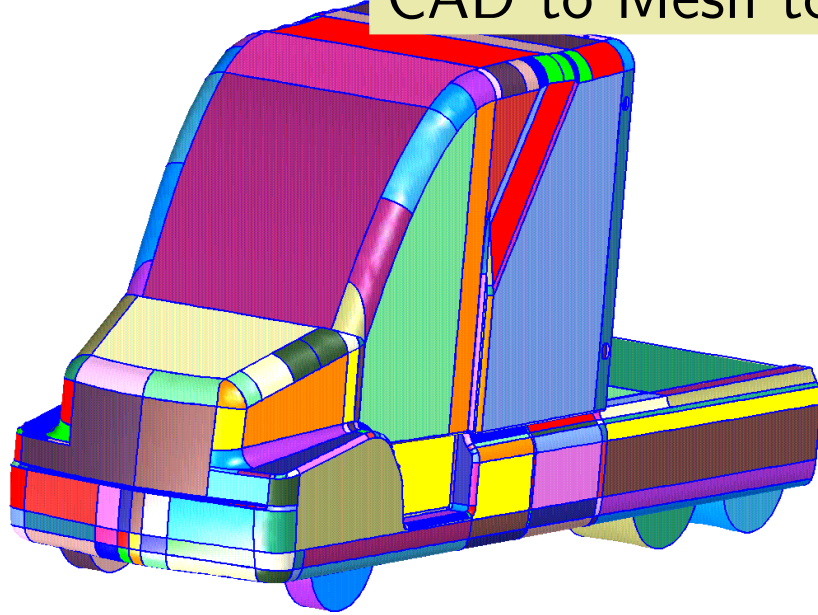
Demo: Creating Mappings...

- show different types of mappings
- composition, stretching, body of revolution...

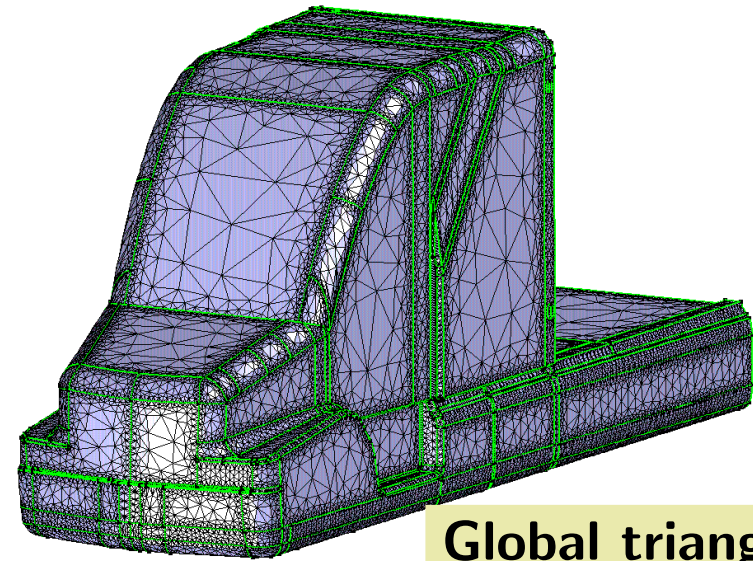
Repairing and Modifying CAD Geometry

- Overture can read the **IGES** CAD file format.
- Geometry is usually defined as a B-Rep (boundary representation) consisting of a collection of patches. Each patch is often a trimmed NURBS (non-uniform rational b-spline).
- There are problems that may exist in the CAD representation such as gaps or overlaps between patches and mistakes in trimming curves.
- The Overture **rap** program (an outcome from the Rapsodi project) can be used to fix and modify the CAD geometry.
- A *global triangulation* is constructed that defines a water-tight surface.
- The CAD B-Rep and global-triangulation are used when building structured surface grids.

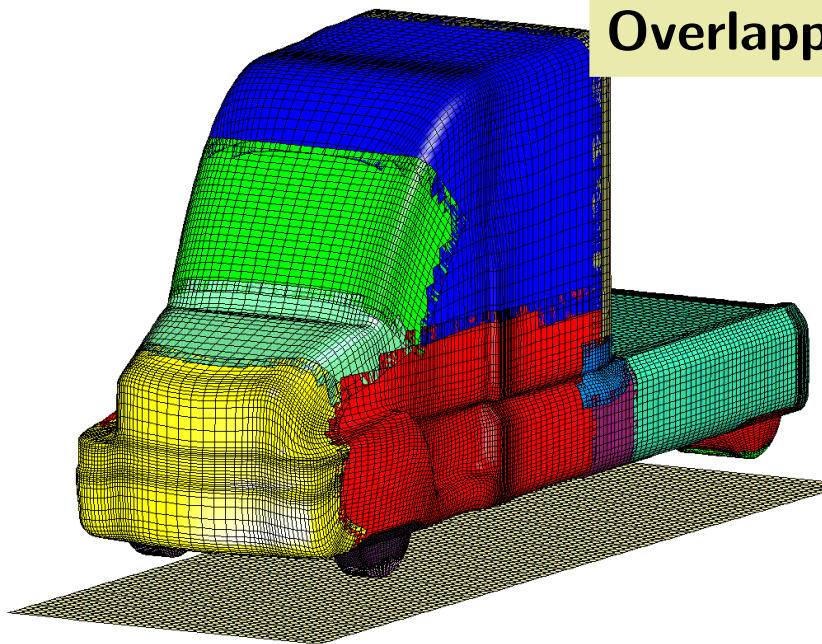
CAD to Mesh to Solution with Overture



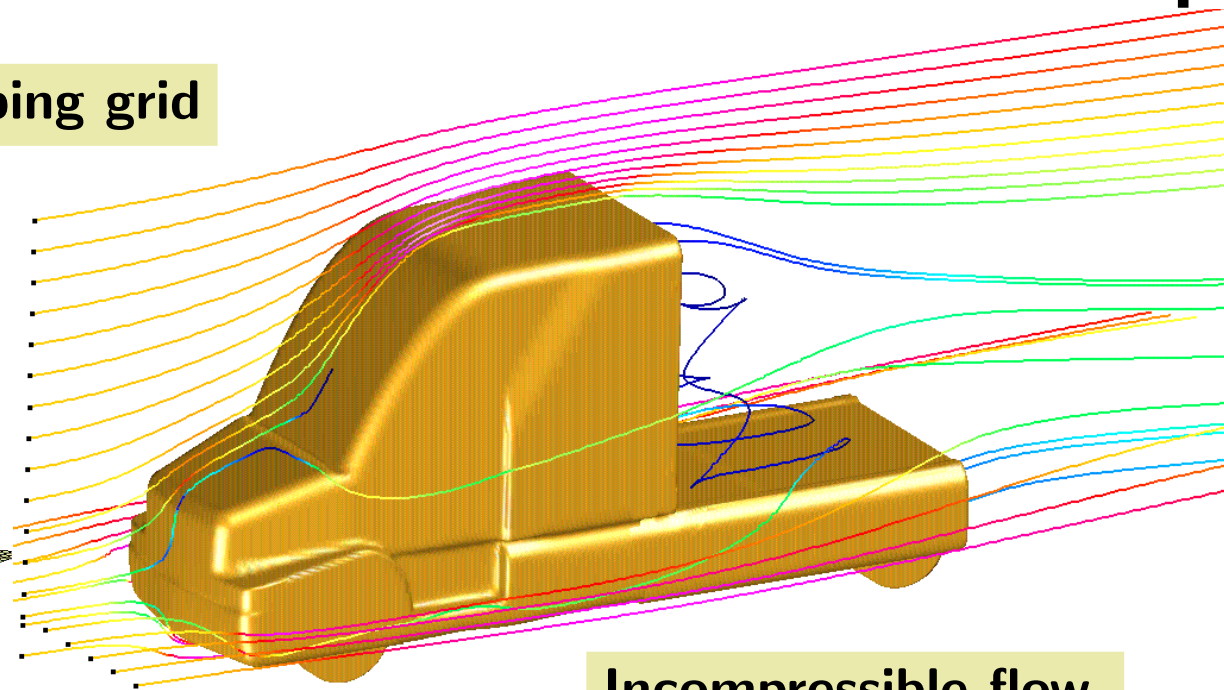
Cad fixup



Global triangulation



Overlapping grid



Incompressible flow.

Demo: Cleaning-up CAD...

Run “**rap asmoNoWheels.cmd**” (in Overture/sampleGrids)

- clean up a CAD geometry for the *asmo* model car.
- generate the topology (fix gaps and overlaps between patches).
- build a water-tight surface triangulation.

Grids on CAD, Ogen...

Constructing Grids on CAD Geometry

- **mBuilder** and **rap** can be used to build grids for CAD geometries.
- Surface grids are constructed first. Volume grids are generated starting from the surface grids.
- Grids are usually built using the hyperbolic grid-generator **hype**.
- Given a starting curve on the surface, a grid is generated by marching over the surface.
- Volume grids are constructed by marching, starting from a surface grid.
- Grids can be smoothed with an elliptic smoother. Grid lines can be clustered (stretched).

Demo: Constructing Grids on CAD geometries...

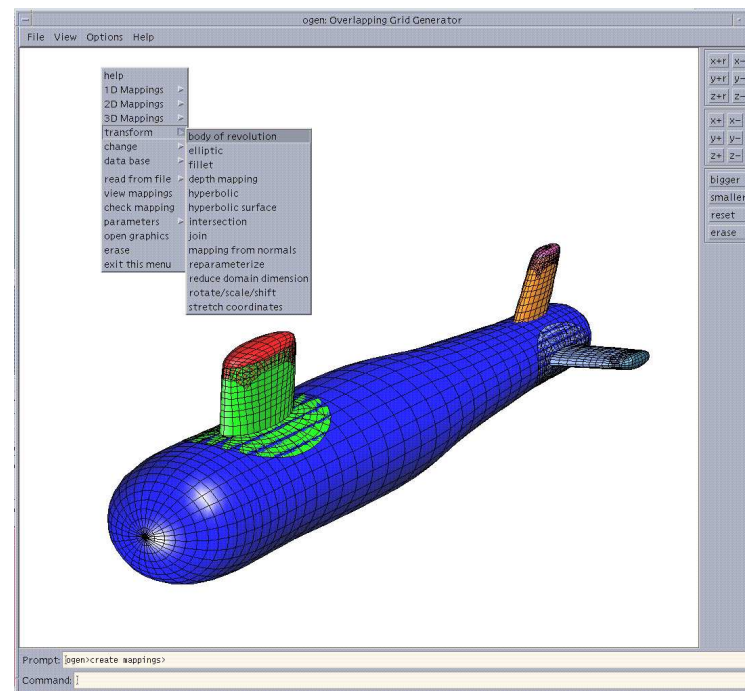
Run “mbuilder asmoBody.cmd”, “mbuilder asmoFrontWheel.cmd”,
“mbuilder asmoBackWheel.cmd” (in Overture/sampleGrids)

- Build grids on the surface of the asmo car (with wheels removed).
- Define a starting curve; build a surface grid by matching; build a volume grid by marching.

Generating Overlapping Grids with Ogen...

The basic steps to follow when creating an overlapping grid are

- create mappings that cover a domain and overlap where they meet.
- generate the overlapping grid (the executable named ogen calls the grid generator Ogen).
- save the grid in a data-base file.



A snapshot of the overlapping grid generator Ogen

Generating Overlapping Grids with Ogen...

- **Ogen** determines the *holes* and interpolation information for an overlapping grid.
- Input to **Ogen** is a set of overlapping grids, boundary conditions and shared-boundary information.
- Options: *discretization width*, *interpolation width* are specified according to the intended equations and order of accuracy.
- Boundaries are classified as physical boundaries, periodic or interpolation.
- Physical boundaries are used to cut holes in overset grids.
- The overlap is usually *minimized* to reduce the number of computational points.

Sample ogen command file...

```
* make a grid for a square
create mappings
  rectangle
  mappingName
    square
  lines
    11 11
  boundary conditions
    1 1 1 1
  exit
exit
*
generate an overlapping grid
  square
  done
  change parameters
    ghost points
    all
    2 2 2 2 2 2
  exit
  compute overlap
exit
*
save an overlapping grid
  square10.hdf
  square10
exit
```


An Ogen command file (script) with embedded perl...

- Any line containing a semi-colon ";" is sent to the perl interpreter and discarded.
- Any line with a "\$" is evaluated by perl to replace variables.

```
*
* Random cylinders in a channel
*
create mappings
*
* Increase $factor to increase the resolution:
$factor=8; $name="multiCylRandom8.hdf";
*
* perl function to convert the
* number of grid points by $factor
sub getGridPoints\
{ local($n1,$n2,$n3)=@_; \
  $nx=int(($n1-1)*$factor+1.5);\
  $ny=int(($n2-1)*$factor+1.5);\
  $nz=int(($n3-1)*$factor+1.5);\
}
*
rectangle
  set corners
    -2. 2. -1.5 1.5
  lines
    getGridPoints(161,121);
```

```
    $nx $ny
    boundary conditions
    1 1 1 1
    mappingName
    backGround
exit
* =====
* Define a perl function to build and AnnulusMapping
* usage: makeAnnulus(radius,xCenter,yCenter,name)
* output: $commands
* =====
sub makeAnnulus\
{ local($radius,$xc,$yc,$name)=@_; \
  $outerRadius=$radius+.15/$factor;\
  $nxr = int($nx*$radius/.25+.5); \
  $annulusMappingNames = $annulusMappingNames .
  $commands = \
  "Annulus\n" . \
  "lines\n" . \
  " $nxr $ny\n" . \
  "inner and outer radii\n" . \
  " $radius $outerRadius\n" . \
```

```

"centre\n" . \
" $xc $yc\n" . \
"boundary conditions\n" . \
" -1 -1 1 0\n" . \
"mappingName\n" . \
" $name\n" . \
"exit\n"; \
}
*
getGridPoints(81,7);
* fix lines in the normal direction
* since we reduce the radius:
$ny=7;
*
*
makeAnnulus(.125,-1.2,0.2,annulus1);
$commands
*
makeAnnulus(.1,-.8,-.5,annulus2);
$commands
*
makeAnnulus(.0625,-.45,.45,annulus3);
$commands
*
... (lines left out here)
*

```

```

makeAnnulus(.13,-.4,-.9,annulus10);
$commands
*
makeAnnulus(.19,-.1,.8,annulus11);
$commands
*
*
exit
generate an overlapping grid
  backGround
  $annulusMappingNames
done
change parameters
  * choose implicit or explicit interpolation
  * interpolation type
  * implicit for all grids
ghost points
  all
  2 2 2 2 2 2
exit
compute overlap
exit
*
save an overlapping grid
  $name
  multiCylRandom
exit

```

Demo: Generating an Overlapping Grid with Ogen...

- circle-in-a-channel example *by hand*.
- Build a grid for a valve, port and cylinder using the native geometry tools. This example illustrates the building of a body-of-revolution and the creation of a special “join” mapping where the valve-stem intersects the port.

Primer...

Primer Examples: Using the High-Level Interface

Overture is distributed with a collection of *Primer* examples that can be used as a starting point for creating new solvers.

- **MappedGrid Examples:** solving problems on single MappedGrid.
 - mappedGridExample2: Solve a convection diffusion equation on a single curvilinear grid.
- **Overlapping Grid Examples**
 - example 6: Solve a convection diffusion equation on an overlapping grid.
 - example 7: Solve an elliptic boundary value problem.

mappedGridExample2: Solving $u_t + au_x + bu_y = \nu \Delta u$. Setup...

```
#include "Overture.h"
...
int
main(int argc, char *argv[])
{
    Overture::start(argc,argv); // initialize Overture
    AnnulusMapping annulus;
    MappedGrid mg(annulus);           // MappedGrid for a square
    mg.update();                     // create default variables

    realMappedGridFunction u(mg);
    Index I1,I2,I3;
    getIndex(mg.dimension(),I1,I2,I3); // assign I1,I2,I3 from dimension
    u(I1,I2,I3)=1.;                  // initial conditions

    MappedGridOperators op(mg);       // operators
    u.setOperators(op);              // associate with a grid function

    PlotStuff ps(TRUE,"mappedGridExample2"); // create a PlotStuff object
    PlotStuffParameters psp;         // This object is used to change plotting parameters
```

mappedGridExample2: $u_t + au_x + bu_y = \nu \Delta u$. Time-stepping...

```
real t=0, dt=.005, a=1., b=1., nu=.1;
for( int step=0; step<100; step++ )
{
    if( step % 10 == 0 )
        PlotIt::contour(ps, u,psp );    // plot contours every 10 steps

    // ***** forward Euler time step *****
    u+=dt*( -a*u.x() -b*u.y() + nu*(u.xx()+u.yy()) );

    t+=dt;

    // apply Boundary conditions : u=0
    int component=0;
    u.applyBoundaryCondition(component,dirichlet,allBoundaries,0.);
    // fix up corners, periodic update:
    u.finishBoundaryConditions();
}

Overture::finish();
return 0;
}
```

Example 6: Solving a PDE on an overlapping grid

This example solves $u_t + au_x + bu_y = \nu(u_{xx} + u_{yy})$ on an overlapping grid.

```
#include "Overture.h"
#include "Ogshow.h"
#include "CompositeGridOperators.h"

int
main(int argc, char *argv[])
{
    Overture::start(argc,argv); // initialize Overture

    aString nameOfOGFile="cic.hdf", nameOfShowFile="example6.show";

    // create and read in a CompositeGrid
    CompositeGrid cg;
    getFromADatabase(cg,nameOfOGFile);
    cg.update();

    Interpolant interpolant(cg); // Make an interpolant

    Ogshow show( nameOfShowFile ); // create a show file

    CompositeGridOperators operators(cg); // operators for a CompositeGrid
    // operators.setOrderOfAccuracy(4); // use this for fourth order

    Range all;
    realCompositeGridFunction u(cg,all,all,all,1); // create a grid function
    u.setOperators(operators);
    u.setName("u"); // name the grid function

    u=1.; // initial condition
```



```

real t=0, dt=.001;                                // initialize time and time step
real a=1., b=1., viscosity=.1;                     // initialize parameters

char buffer[80];                                    // buffer for sprintf
int numberOfTimeSteps=200;
for( int i=0; i<numberOfTimeSteps; i++ )           // take some time steps
{
    if( i % 40 == 0 ) // save solution every 40 steps
    {
        show.startFrame();                          // start a new frame
        show.saveComment(0,sPrintf(buffer,"Here is solution %i",i));
        show.saveComment(1,sPrintf(buffer,"  t=%e ",t));
        show.saveSolution( u );
    }

    // *** take a time step with Euler's method ***
    u+=dt*( -a*u.x() - b*u.y() + viscosity*(u.xx() + u.yy()));

    t+=dt;
    u.interpolate();                                // interpolate
    // apply a dirichlet BC on all boundaries:
    u.applyBoundaryCondition(0,BCTypes::dirichlet,BCTypes::allBoundaries,0.);
    // for 4th order:
    // u.applyBoundaryCondition(0,BCTypes::extrapolate,BCTypes::allBoundaries,0.);
    u.finishBoundaryConditions();
}

Overture::finish();
return 0;
}

```

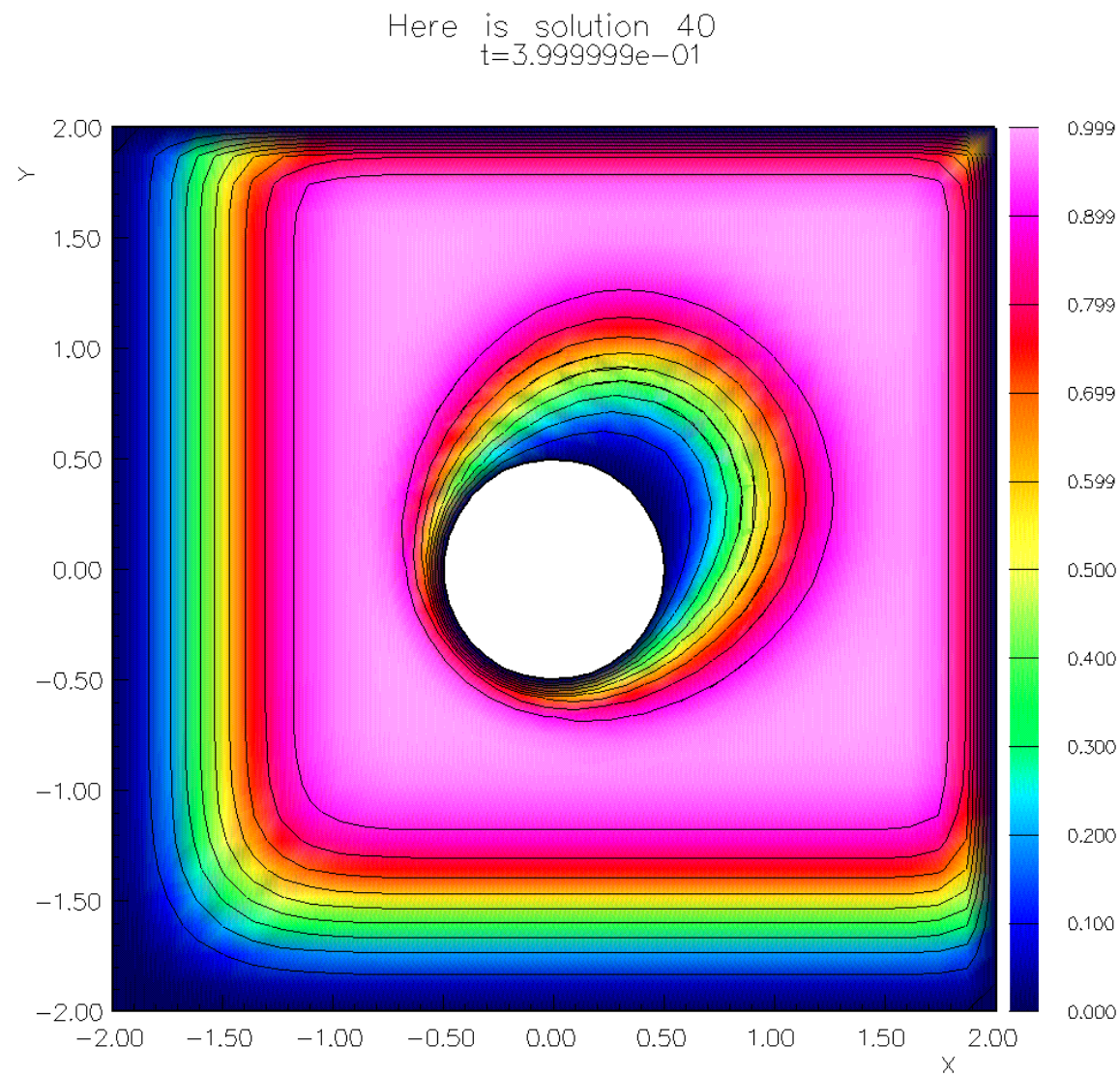


Figure 1: Results from example6.C, solve an advection-diffusion equation.

Example 7: Solve an elliptic boundary value problem

This example solves $\Delta u + u_x = f$ with Dirichlet BC's on an overlapping grid.

```
#include "Overture.h"
#include "CompositeGridOperators.h"
#include "Oges.h"

int main(int argc, char *argv[])
{
    Overture::start(argc,argv); // initialize Overture
    // create and read in a CompositeGrid
    aString nameOfOGFile="cic.hdf";
    CompositeGrid cg;
    getFromADatabase(cg,nameOfOGFile);
    cg.update();

    // make a grid function to hold the coefficients
    Range all;
    int stencilSize=int( pow(3,cg.numberofDimensions()+1.5 ));
    realCompositeGridFunction coeff(cg,stencilSize,all,all,all);
    coeff.setIsACoefficientMatrix(TRUE,stencilSize);

    realCompositeGridFunction u(cg),f(cg); // create grid functions:

    CompositeGridOperators op(cg); // create some differential operators
    op.setStencilSize(stencilSize);
    coeff.setOperators(op);

    coeff=op.laplacianCoefficients()+op.xCoefficients(); // here is the operator
    // fill in the coefficients for the boundary conditions
```

```

coeff.applyBoundaryConditionCoefficients(0,0,dirichlet, allBoundaries);
coeff.applyBoundaryConditionCoefficients(0,0,extrapolate,allBoundaries);
coeff.finishBoundaryConditions();

Oges solver( cg );          // create a solver
solver.setCoefficientArray( coeff );    // supply coefficients

// assign the rhs: u=0 on the boundary
Index I1,I2,I3, Ib1,Ib2,Ib3;
for( int grid=0; grid<cg.numberOfComponentGrids(); grid++ ){
    MappedGrid & mg = cg[grid];
    getIndex(mg.indexRange(),I1,I2,I3);
    f[grid](I1,I2,I3)=1.;
    for( int side=Start; side<=End; side++ )
        for( int axis=axis1; axis<cg.numberOfDimensions(); axis++ ){
            if( mg.boundaryCondition()(side,axis) > 0 ){
                getBoundaryIndex(mg.gridIndexRange(),side,axis,Ib1,Ib2,Ib3);
                f[grid](Ib1,Ib2,Ib3)=0.;
            }
        }
}

solver.solve( u,f );    // solve the equations
u.display("Here is the solution");

Overture::finish();
return(0);
}

```

Demo: Primer examples...

- mappedGridExample2: solve a PDE on a mappedGrid.
- move1: moving grid example.

AMR...

Block Structured Adaptive Mesh Refinement

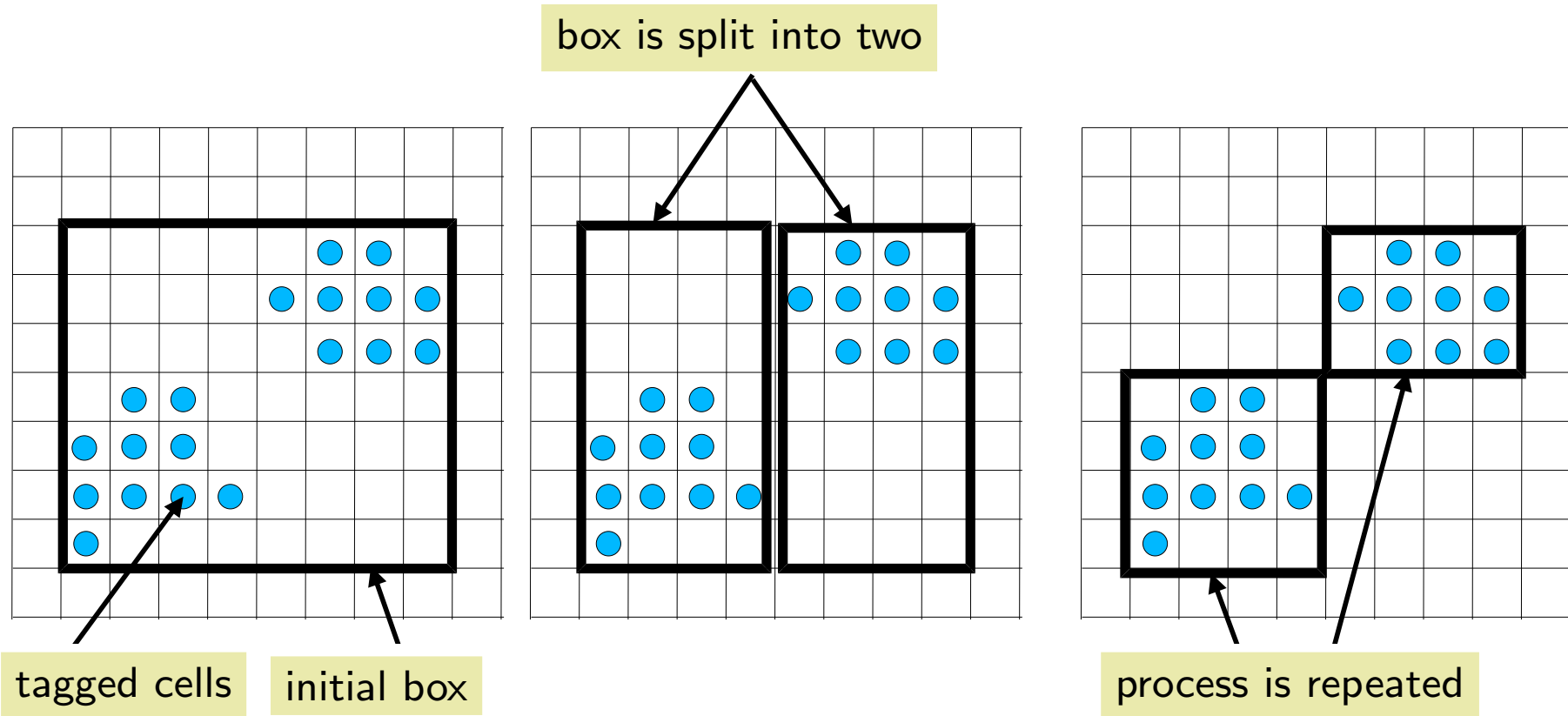
- ◇ Initially developed by Berger and Oliger (JCP 1984)
- ◇ Extensions to the Euler equations by Berger and Colella (JCP 1989)
- ◇ AMR and overlapping grids considered by Brislawn, Brown, Chesshire and Saltzman (1995), and Boden and Toro (1997)
- ◇ AMR in Overture has contributions from Brown, Philip and Quinlan.

Some Structured AMR frameworks

- ◇ AMRCLAW (LeVeque and Berger)
- ◇ Amrita (Quirk)
- ◇ Boxlib (Bell et.al., LBNL)
- ◇ Chombo (Colella et.al., LBNL)
- ◇ GrACE (Parashar)
- ◇ PARAMESH (NASA Goddard Space Flight Center)
- ◇ SAMRAI (Hornung et.al. LLNL)

Reference Tomasz Plewa's AMR page <http://flash.uchicago.edu/~tomek/AMR>

AMR regridding algorithm (Berger-Rigoutsos)



- (1) tag cells where refinement is needed
- (2) create a box to enclose tagged cells
- (3) split the box along its long direction based on a histogram of tagged cells
- (4) fit new boxes to each split box and repeat the steps as needed.

AMR on overlapping grids

AMR capabilities are being added to Overture for overlapping grids.

Some of the features are

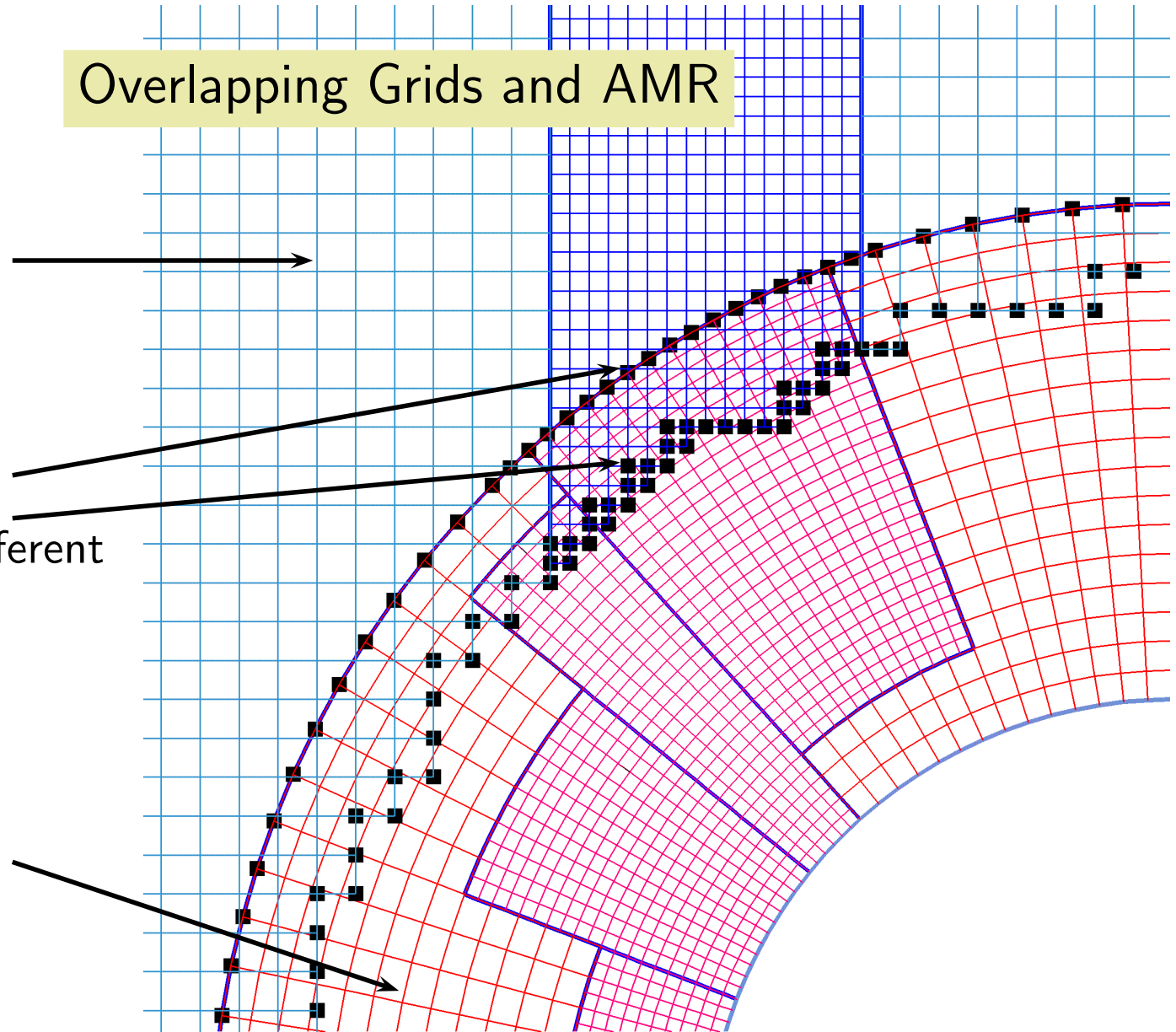
- AMR grids are generated in the unit square coordinates of each component grid.
- efficient handling of refinement grids on curvilinear grids.
- support for higher order accurate methods (fourth-order, sixth-order,...) (not complete yet)
- updating refinement grids that meet at the overlapping grid boundaries.
- retaining the efficiency of Cartesian grids.
- saving and reading solutions and grids from a data base file in an efficient manner (e.g. for post-processing and restarts).
- interactive graphics.

Overlapping Grids and AMR

Component grid 1,
base grid 1

Refinement grids
interpolate from
refinements of a different
base grid

Component grid 2,
base grid 2



The basic AMR time stepping algorithm

PDEsolve($\mathcal{G}, t_{\text{final}}$)

{ \mathcal{G} (input): current grid

$t := 0; n := 0;$

$\mathbf{u}_i^n := \text{applyInitialCondition}(\mathcal{G});$

while $t < t_{\text{final}}$

if ($n \bmod n_{\text{regrid}} == 0$)

$e_i := \text{estimateError}(\mathcal{G}, \mathbf{u}_i^n);$

$\mathcal{G}^* := \text{regrid}(\mathcal{G}, e_i);$

$\mathbf{u}_i^* := \text{interpolateToNewGrid}(\mathbf{u}_i^n, \mathcal{G}, \mathcal{G}^*);$

$\mathcal{G} := \mathcal{G}^*; \mathbf{u}_i^n := \mathbf{u}_i^*;$

end

$\Delta t := \text{computeTimeStep}(\mathcal{G}, \mathbf{u}_i^n);$

$\mathbf{u}_i^{n+1} := \text{timeStep}(\mathcal{G}, \mathbf{u}_i^n, \Delta t);$

$t := t + \Delta t; n := n + 1;$

$\text{interpolate}(\mathcal{G}, \mathbf{u}_i^n);$

$\text{applyBoundaryConditions}(\mathcal{G}, \mathbf{u}_i^n, t);$

end

}

AMR components of Overture

class Regrid

- generation of aligned AMR grids using the Berger-Rigoutsos algorithm.
- generation of rotated AMR grids using the Berger algorithm.
- Boxlib is used for domain calculus (e.g. intersecting two boxes).

class ErrorEstimator

- defines standard error estimators based on first and second differences.
- smoothing of the error and propagation across overlapping grid boundaries.

class Interpolate

- fine to coarse and coarse to fine interpolation of patches.
- supports *any* refinement ratio (1,2,3,4,...) and *any* order of accuracy.

AMR components of Overture

class InterpolateRefinements

- high level function to interpolate the solution from one AMR overlapping grid to another AMR overlapping grid.
- update all AMR ghost points and hidden coarse grid points on an AMR overlapping grid.

Ogen

- the overlapping grid generator knows how to update interpolation points on AMR grids.

Elliptic Solvers

- Oges can be used to solve for the solution on the entire AMR hierarchy. Oges is an interface to sparse solvers such as PETSc.
- Ogmng multigrid solver will be extended to AMR, work with Bobby Philip (LANL).

AMR Performance on two problems:

	Quarter plane		Expanding channel	
time steps	12,418		21,030	
grids (min,ave,max)	(2, 57, 353)		(5, 274, 588)	
points (min,ave,max)	(2.0e5, 9.2e5, 1.9e6)		(1.2e5, 6.4e5, 1.3e6)	
	s/step	%	s/step	%
compute $\Delta u_{i,j}^n$	13.85	92.7	11.50	82.4
boundary conditions	.12	.8	.14	1.0
interpolation (overlapping)	.09	.6	.45	3.2
AMR regrid/interpolation	.54	3.6	1.62	11.6
other	.34	2.3	.25	1.8
total	14.94	100	13.96	100

Table 2: CPU time (in seconds) per step for various parts of the code and their percentage of the total CPU time per step.

amrHype: Solve a convection diffusion equation with AMR

- a small code demonstrating the use of AMR with Overture.
- uses the *method of analytic solutions* for testing accuracy.

Sample uses of Overture AMR functions:

```
CompositeGrid cg,cgNew;  
realCompositeGridFunction u,error,uNew;  
ErrorEstimator errorEstimator;  
    errorEstimator.computeAndSmoothErrorFunction(u,error);  
Regrid regrid;  
    regrid.regrid(cg,cgNew, error, errorThreshold );  
Ogen ogen;  
    ogen.updateRefinement(cgNew);  
uNew.updateToMatchGrid(cgNew);  
InterpolateRefinements interp;  
    interp.interpolateRefinements( u,uNew );
```

Testing using the method of analytic solutions

The usefulness of this technique cannot be overstated.

Given a PDE boundary value problem

$$L(u_t, u_x, u_y, \dots) = F(\mathbf{x}, t)$$

one can create an *exact* solution, $U(\mathbf{x}, t)$ by choosing

$$F(\mathbf{x}, t) = L(U_t, U_x, U_y, \dots)$$

The Overture OGFunction class defines a variety of exact solutions and their derivatives to support the method of analytic solutions. For example one could define a polynomial, trigonometric polynomial, or *pulse function*

$$U(\mathbf{x}, t) = (x^2 + 2xy + y^2 + z^2)(1 + \frac{1}{2}t + \frac{1}{3}t^2)$$

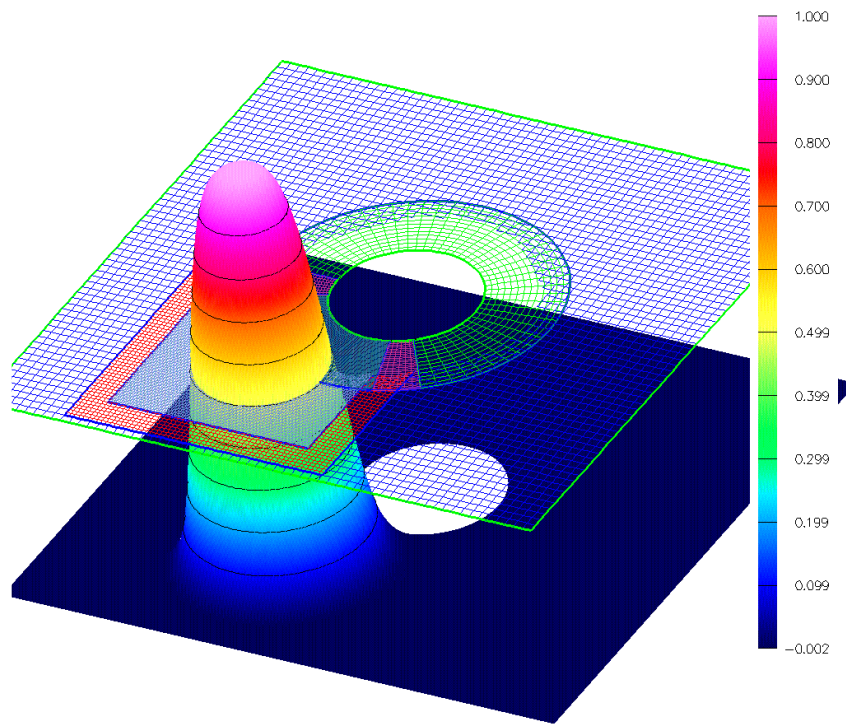
$$U(\mathbf{x}, t) = \cos(\pi\omega x) \cos(\pi\omega y) \cos(\pi\omega z) \cos(\omega_3\pi t)$$

$$U(\mathbf{x}, t) = a_0 \exp(-a_1 \|\mathbf{x} - \mathbf{b}(t)\|^{2p}) , \quad \mathbf{b}(t) = \mathbf{c}_0 + \mathbf{v}t$$

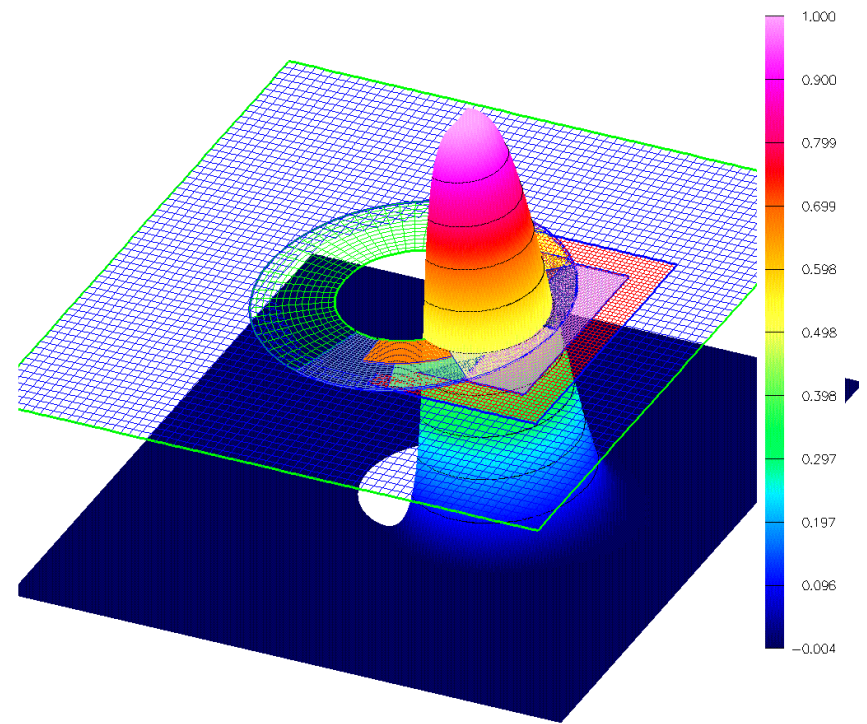
The polynomial solution is particularly useful since this solution is often an exact solution to the discrete equations on rectangular grids. The pulse function is good for AMR.

amrHype: Solve a convection diffusion equation with AMR

u t=5.00e-01, dt=3.78e-03, nu=1.00e-02, anu=0.00e+00

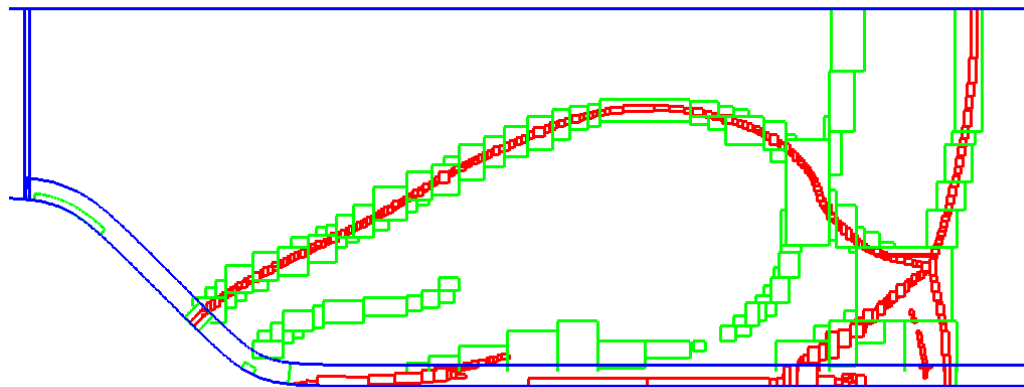
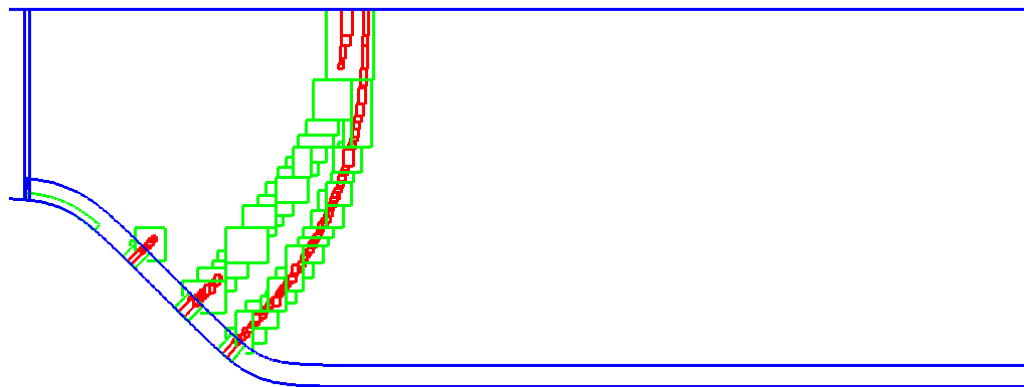
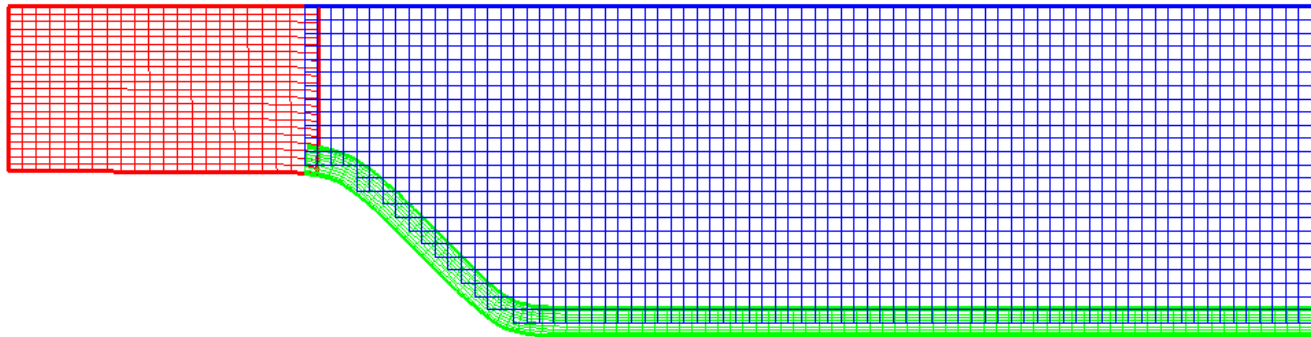


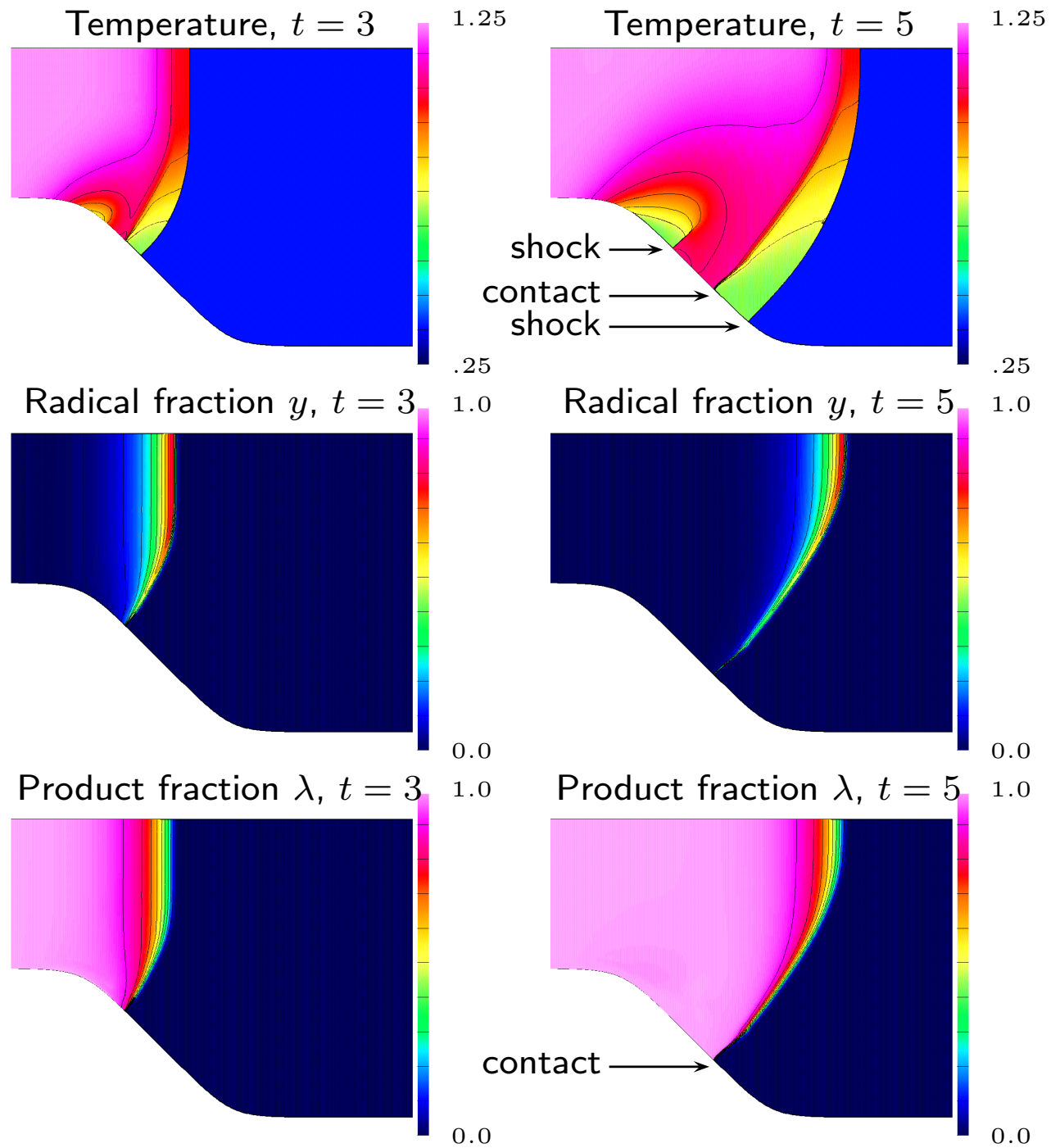
u t=1.50e+00, dt=3.77e-03, nu=1.00e-02, anu=0.00e+00



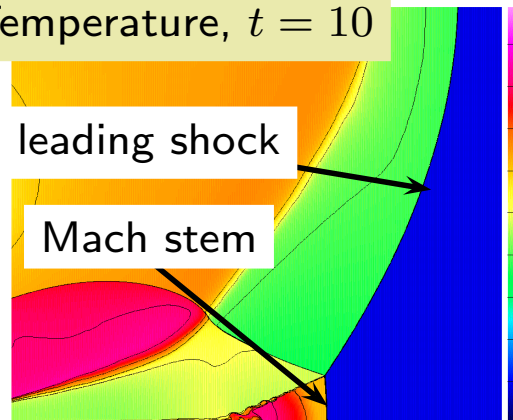
Traveling pulse analytic solution

Detonation in an Expanding Channel.



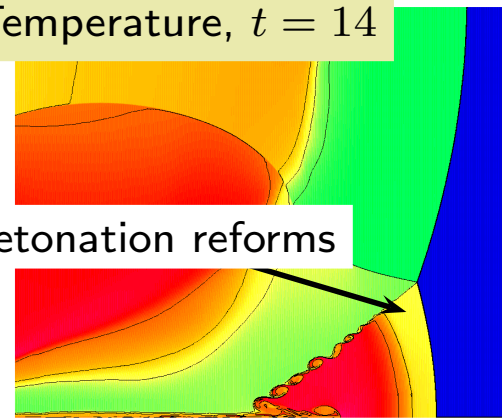


Temperature, $t = 10$



1.5

Temperature, $t = 14$

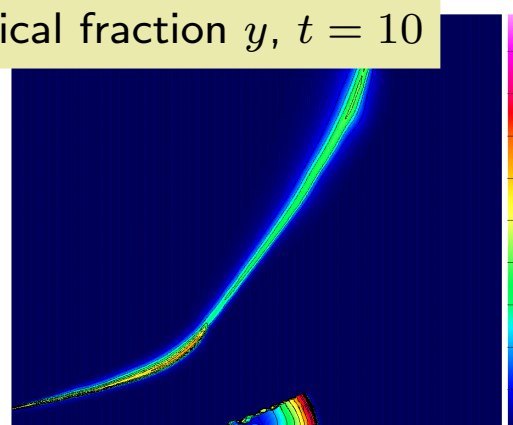


1.25

.25

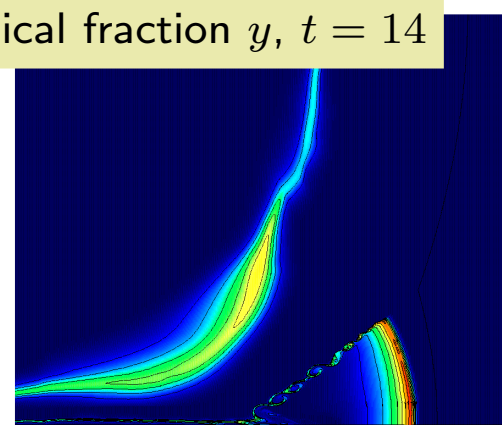
.25

Radical fraction y , $t = 10$



1.0

Radical fraction y , $t = 14$

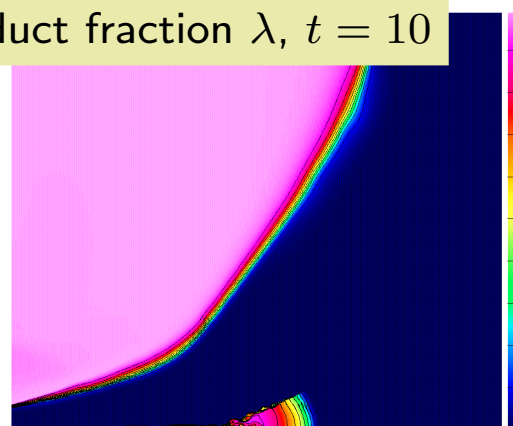


1.0

0.0

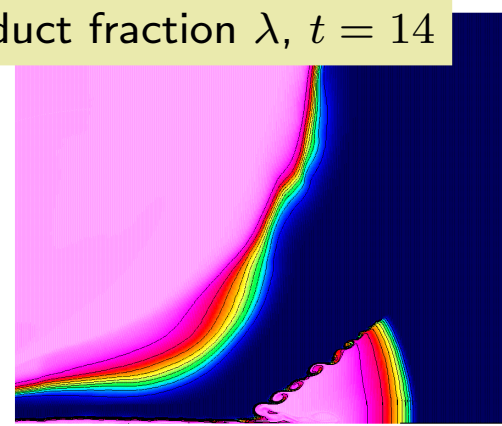
0.0

Product fraction λ , $t = 10$



1.0

Product fraction λ , $t = 14$



1.0

0.0

0.0

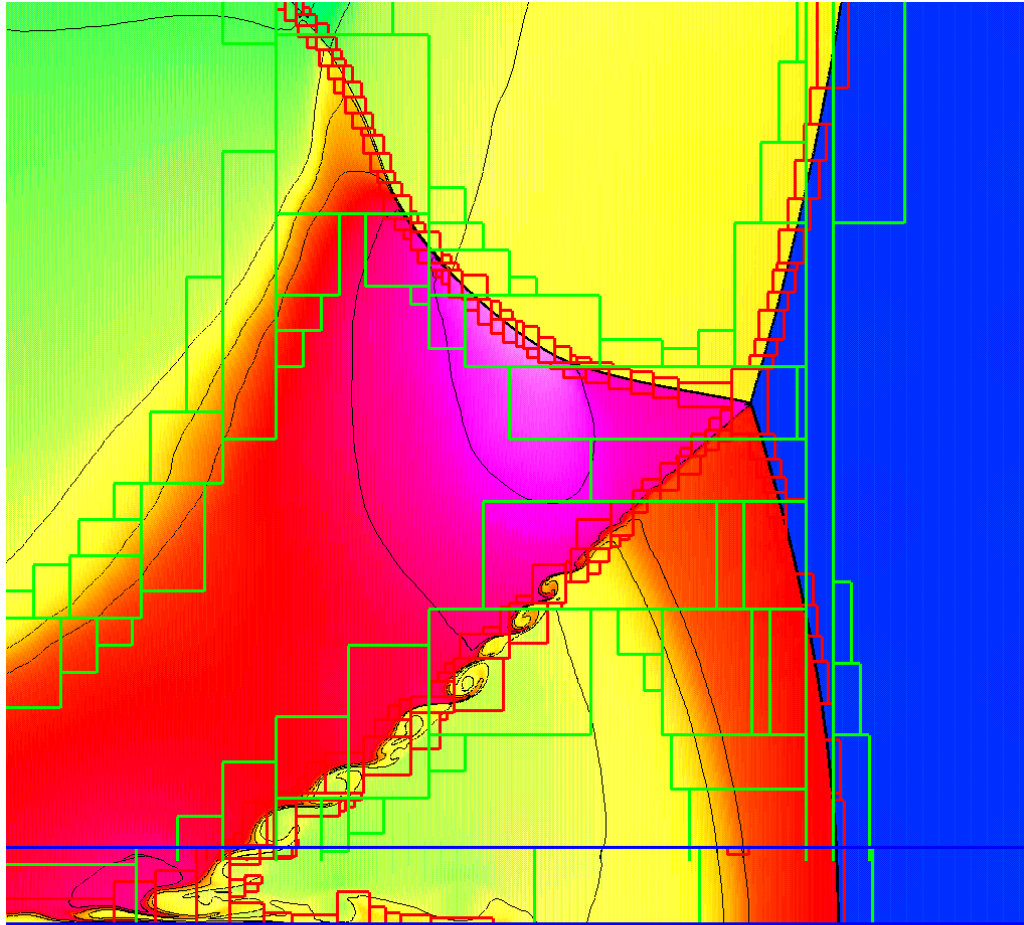


Figure 2: Closeup of the density near the Mach stem. The boundaries of the refinement grids are shown.

Solvers...

Oges: Overlapping Grid Equation Solver

The Oges class can be used to solve the systems of sparse equations that result from discretising boundary value problems (elliptic problems, implicit time-stepping) on overlapping grids.

Oges is an interface to various solvers packages:

- ◇ sparse direct solvers (e.g. Yale)
- ◇ sparse iterative solvers (SLAP, PETSc)
- ◇ multigrid solver (Ogmg)

The primer example7 shows the use of Oges:

```
...
realCompositeGridFunction coeff(cg,stencilSize,all,all,all);
CompositeGridOperators op(cg);           // create some differential operators
...
coeff=op.laplacianCoefficients()+op.xCoefficients(); // here is the operator
...
Oges solver( cg );                      // create an Oges solver
solver.setCoefficientArray( coeff );    // supply coefficients
...
solver.solve( u,f );
```

Here the sparse matrix for the operator $\text{coeff} = \Delta + \partial_x$ is formed.

Ogmg: Overlapping Grid Multigrid Solver

The Ogmg solver can be used to solve scalar elliptic boundary value problems. Some of its key features are

- ◇ coarse grid generation - robust generation of “any” number of multigrid levels.
- ◇ smoothers
 - ◇ variable sub-smooths per component grid
 - ◇ interpolation-boundary smoothing
 - ◇ over-relaxed Red-Black smoothers
- ◇ Galerkin coarse grid operators
- ◇ fourth-order accuracy
- ◇ numerical boundary conditions for Dirichlet and Neumann/mixed problems

The Multigrid method is potentially a near optimal approach

Some care is required to achieve *text-book* convergence rates for overlapping grid problems.

Method	# operations in 2D
Gaussian elimination	$\mathcal{O}(N^3)$
Gaussian elimination (band version)	$\mathcal{O}(N^2)$
Jacobi iteration	$\mathcal{O}(N^2 \log \epsilon)$
Gauss-Seidel iteration	$\mathcal{O}(N^2 \log \epsilon)$
Successive over-relaxation (SOR)	$\mathcal{O}(N^{3/2} \log \epsilon)$
Conjugate gradient (CG)	$\mathcal{O}(N^{3/2} \log \epsilon)$
Multigrid (iterative)	$\mathcal{O}(N \log \epsilon)$
Multigrid (full-multigrid)	$\mathcal{O}(N)$

Complexity of different solution approaches for a 2D Poisson problem, from Trottenberg, Oosterlee and Schüller [17]. The number of unknowns is N and the convergence tolerance is ϵ .

Multigrid and Overlapping Grids: Background

The first overlapping grid computations were apparently performed by G. Starius, a student of Heinz-Otto Kreiss, who solved elliptic [13](1977) and hyperbolic [14](1980) problems.

The first MG solver for overlapping grids seems to be the work of J. Linden reported in Stüben and Trottenberg [16](1982) who showed results for a model problem.

Chesshire and Henshaw [5, 7](1985) extended the CMPGRD overlapping grid generator [1] to generate multigrid levels for general two-dimensional domains. These grids were used to solve elliptic problems in two dimensions for general domains and showed good multigrid convergence rates.

Due to the difficulty in generating the interpolation equations to couple the equations on the coarse grids, most if not all other researchers have left the coarse grids uncoupled, applying a zero Dirichlet or Neumann type boundary condition at interpolation points, Tu & Fuchs [18, 19], Hinatsu & Ferziger [11], Zang & Street [20]. This approach has been called **incomplete multigrid** (ICMG) by Hinatsu & Ferziger. In general it would seem that ICMG can converge no better than an overlapping Schwartz iteration with a convergence rate $1 - O(\delta)$ where δ is the relative width of the overlap.

Multigrid Operators

The multigrid algorithm for overlapping grids remains basically the same as for a single grid. Introduce the following operators

S_h : the composite smoothing operator, an iteration that approximately solves the equation and is effective at reducing the high frequency components of the error.

I_h^H : restriction operator, the operator that transfers a grid function from the fine grid to the coarse grid.

I_H^h : prolongation operator, the operator that transfers a grid function from the coarse grid to the fine grid.

Multigrid Algorithm

while *not converged* **do**

smooth ν_1 times

$$v_h \leftarrow \mathbf{S}_h^{\nu_1} v_h$$

form the defect and transfer to the coarser grid

$$f_H \leftarrow \mathbf{I}_h^H (f_h - L_h v_h)$$

“solve” the defect equation

$$A_H v_H \approx b_H$$

correct the fine grid solution from the coarse grid solution

$$v_h \leftarrow v_h + \mathbf{I}_H^h v_H$$

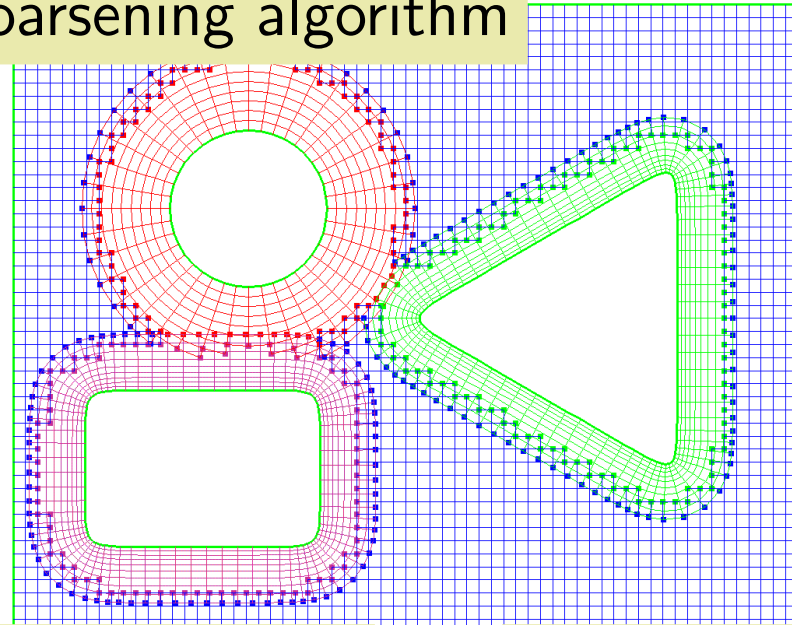
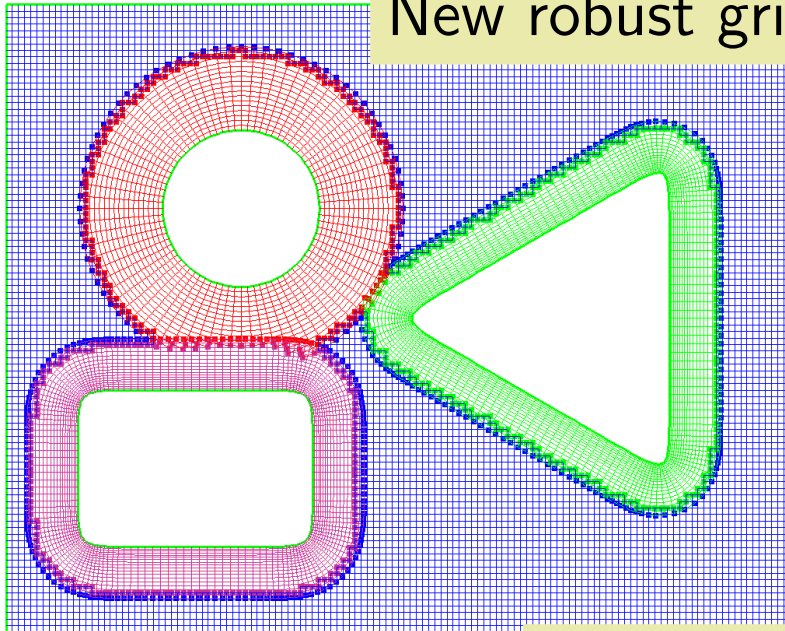
smooth ν_2 times

$$v_h \leftarrow \mathbf{S}_h^{\nu_2} v_h$$

end while

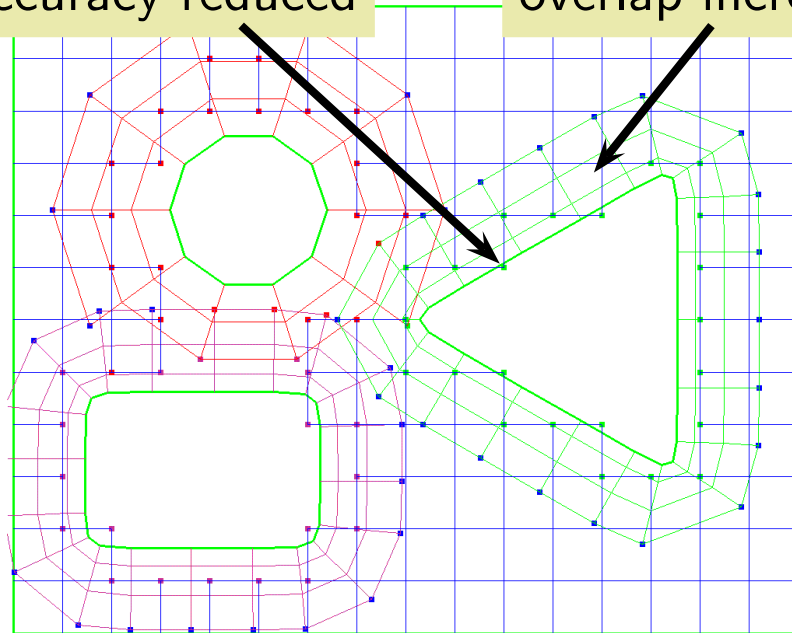
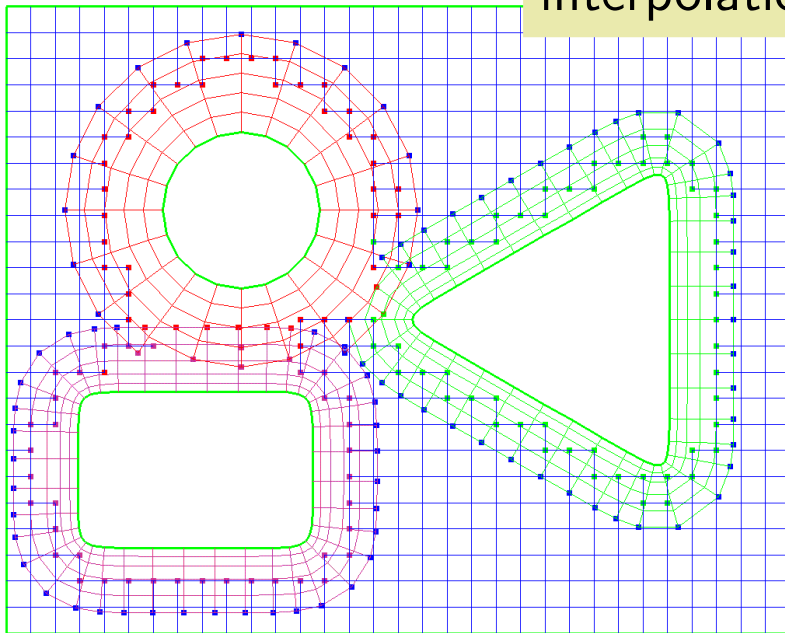
The coarse grid equations can be approximately solved in a recursive many by using an even coarser grid. On the very coarsest grid the equations are solved with a sparse matrix solver using either an iterative or direct method.

New robust grid coarsening algorithm



interpolation accuracy reduced

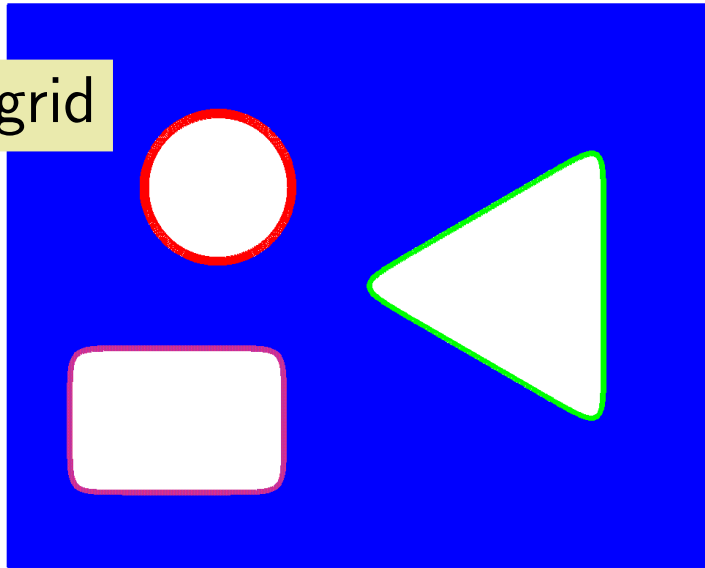
overlap increases



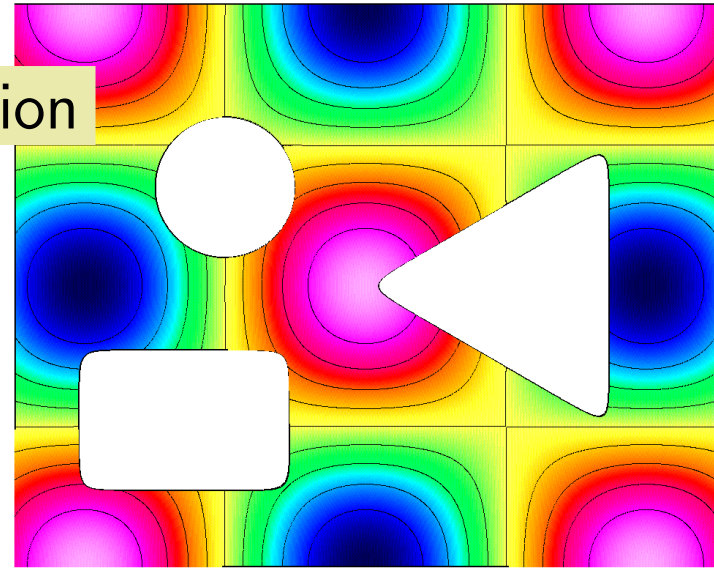
An overlapping grid for some shapes, 4 multigrid levels.

Multigrid solution to Poisson's equation, 3.4 million grid points

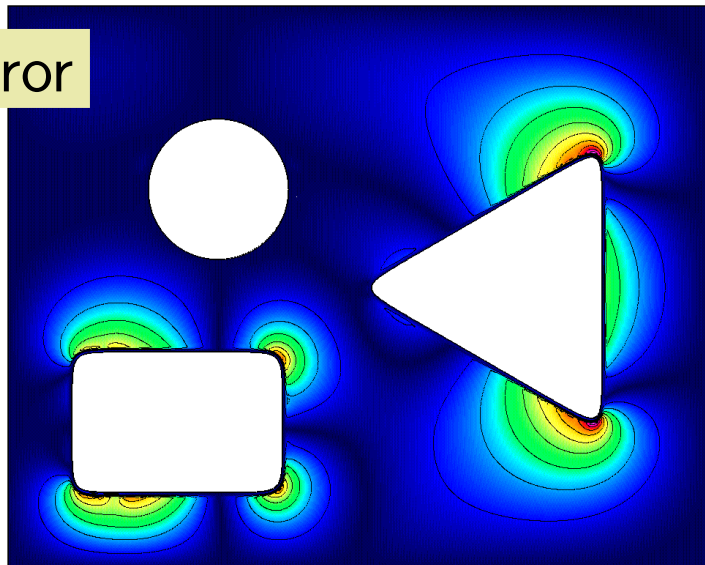
grid



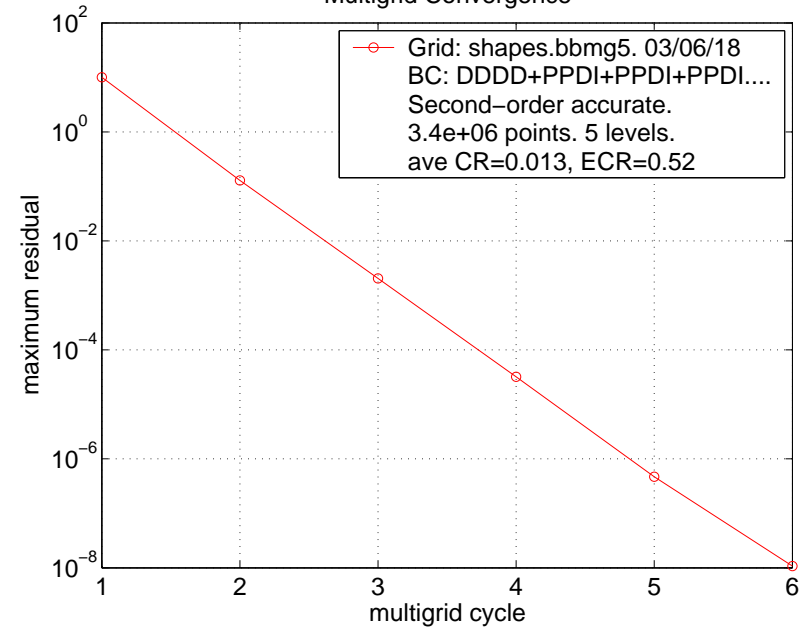
solution



Error

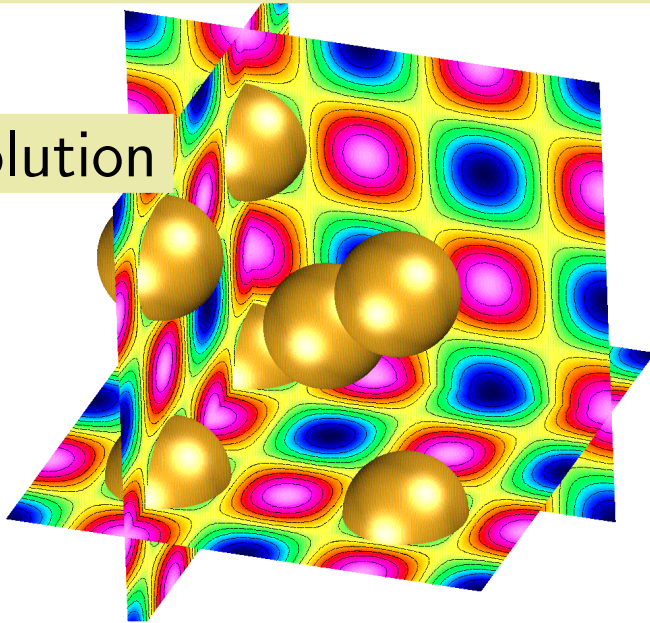


Multigrid Convergence

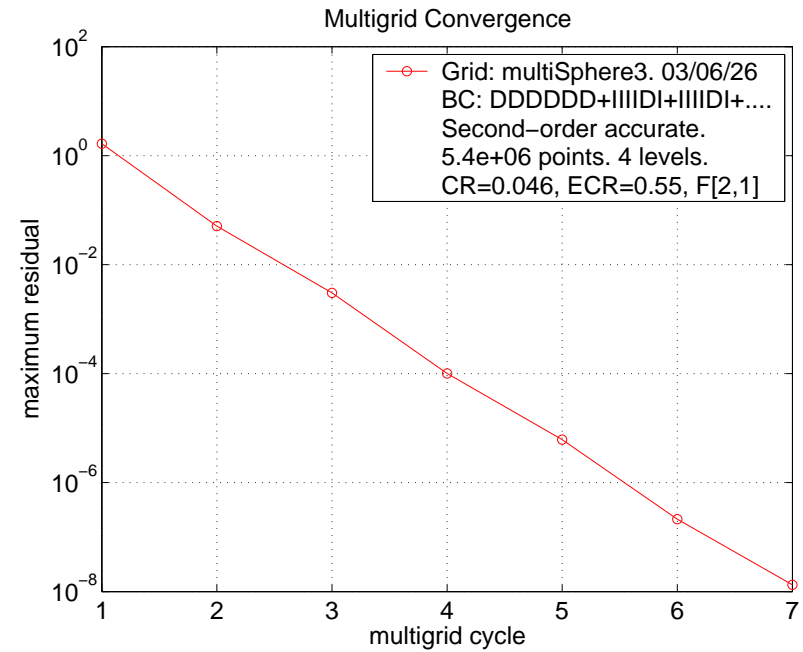
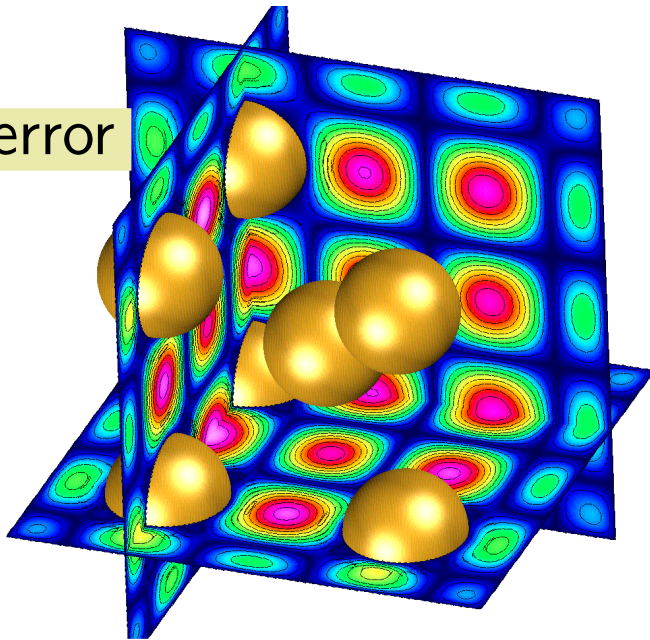


Multigrid solution to Poisson's equation, 5.4 million grid points

solution



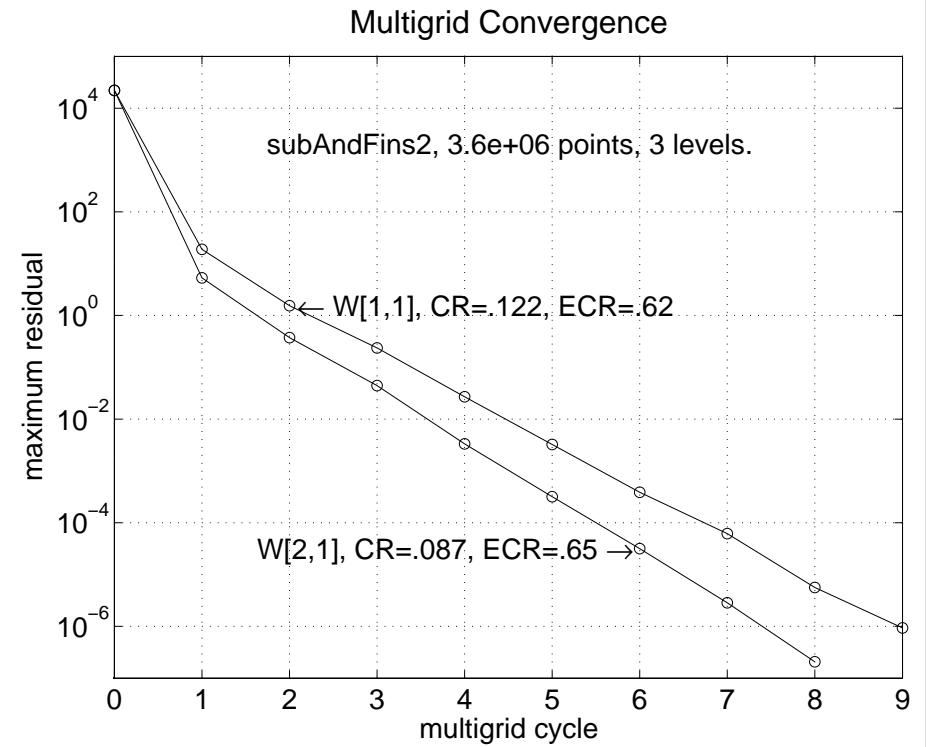
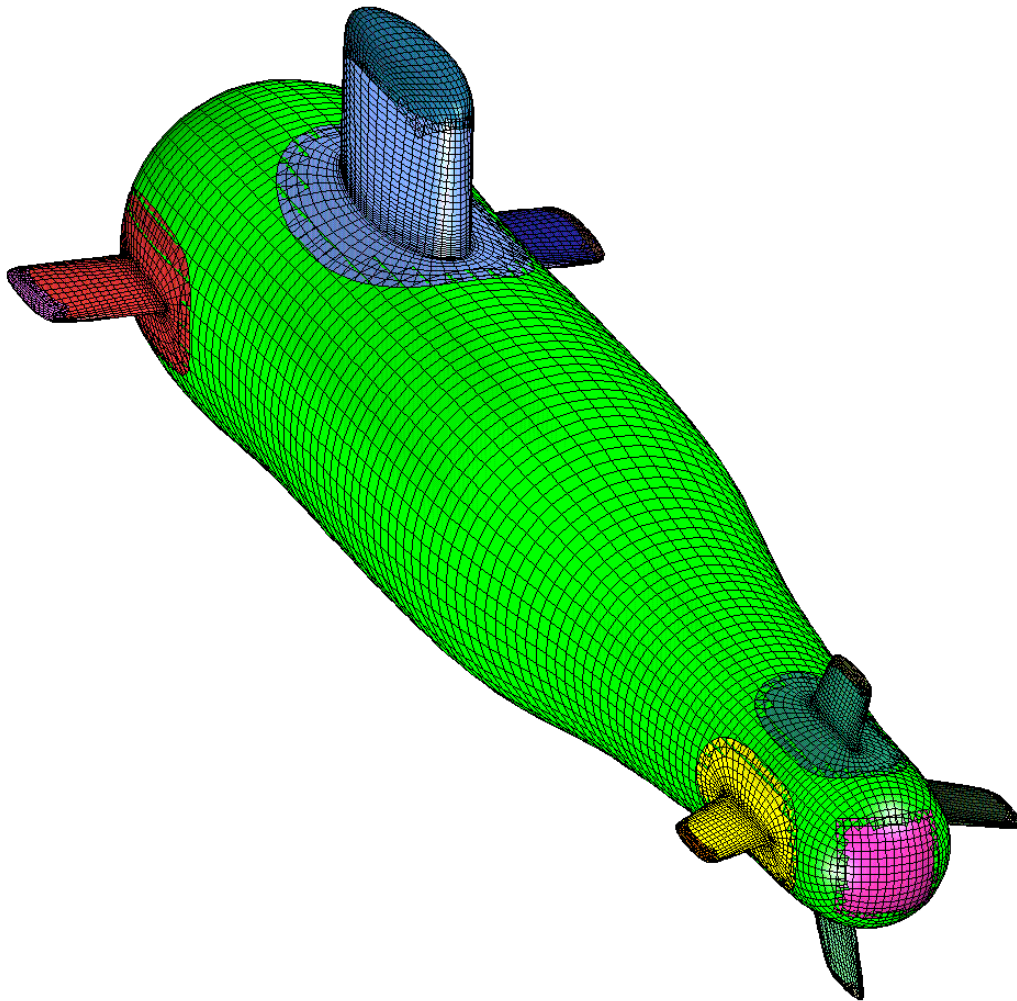
error



Ogmg: multigrid solver for overlapping grids, solving Poisson's equation

					CPU time (s)			storage
Solver	grid	pts	its	$ \text{res} _\infty$	total	setup	solve	reals/pt
Ogmg V[1,1] FMG	cic	1.1e6	7	5.7e-10	3.3	.54	2.8	4.6
biCG-stab, ILU(5)	cic	1.1e6	144	8.9e-9	152	35.	117	53.5
gmres ILU(5)	cic	1.1e6	435	1.0e-8	271	35	236	65.0
biCG-stab, ILU(0)	cic	1.1e6	554	8.6e-9	342	32	310	33.3
gmres ILU(0)	cic	1.1e6	2657	8.9e-9	1135	33	1102	49.0
Ogmg V[1,1] FMG	elb	2.0e6	10	3.3e-10	21.5	4.52	17.0	9.9
biCG-stab, ILU(2)	elb	2.0e6	46	4.1e-10	222.	106	116	70.3
biCG-stab, ILU(0)	elb	2.0e6	113	3.7e-10	264.	77.	187	41.6
gmres(20), ILU(0)	elb	2.0e6	218	5.2e-10	306.	70.	236	56.5

Table 3: A comparison of the multigrid solver Ogmg to some Krylov based solvers (PETSc). The cic grid is a two-dimensional circle-in-a-channel, the elb grid is ellipsoid-in-a-box.



Left: Grid for a submarine-in-a-box, Right: convergence history.

CG: Composite-Grid Solvers

Cgins: Incompressible Navier-Stokes solver

- second and fourth-order order time-accurate solver
- pressure-Poisson formulation
- line-implicit pseudo-steady state solver (local Δt)
- moving grids

Cgcns: Compressible Navier-Stokes solver

- Jameson-style artificial diffusion (with Don Schwendeman, RPI)
- moving grids and AMR

Cgcns: Reactive Euler-equation solver

- High-order Godunov method (Don Schwendeman, RPI)
- moving grids and AMR (in 2D)
- A few reaction mechanisms: one-step, chain branching, ignition-and-growth. and a few equations of state; ideal Gas, JWL.
- A multi-fluid formulation (Jeff Banks, RPI).

The **incompressible Navier-Stokes equations** in velocity-divergence form are

$$\begin{aligned}\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p &= \nu \Delta \mathbf{u}, \\ \nabla \cdot \mathbf{u} &= 0.\end{aligned}$$

In cghs the equations are solved in the velocity-pressure form

$$\begin{aligned}\left. \begin{aligned}\mathbf{u}_t + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p - \nu \Delta \mathbf{u} - \mathbf{f} &= 0 \\ \Delta p - (\nabla u \cdot \mathbf{u}_x + \nabla v \cdot \mathbf{u}_y + \nabla w \cdot \mathbf{u}_z) - C_d(\nu) \nabla \cdot \mathbf{u} - \nabla \cdot \mathbf{f} &= 0\end{aligned}\right\} & \mathbf{x} \in \Omega \\ \left. \begin{aligned}B(\mathbf{u}, p) &= 0 \\ \nabla \cdot \mathbf{u} &= 0\end{aligned}\right\} & \mathbf{x} \in \partial\Omega \\ \mathbf{u}(\mathbf{x}, 0) &= \mathbf{u}_0(\mathbf{x}) & \text{at } t = 0\end{aligned}$$

There are d boundary conditions, $B(\mathbf{u}, p) = 0$, in d space dimensions. On a no-slip wall, for example, $\mathbf{u} = 0$. The boundary condition $\nabla \cdot \mathbf{u} = 0$ is added as a *boundary condition for the pressure*. With this extra boundary condition it follows that the above problem is equivalent to the velocity-divergence formulation. The term $C_d(\nu) \nabla \cdot \mathbf{u}$ is used to damp the divergence.

Cgins: discretization of the incompressible Navier-Stokes equations

Let \mathbf{V}_i and P_i denote the discrete approximations to \mathbf{u} and p so that

$$\mathbf{V}_i \approx \mathbf{u}(\mathbf{x}_i) \quad , \quad P_i \approx p(\mathbf{x}_i) \quad .$$

After discretizing in space (using the mapping-method described earlier) the equations take the form

$$\left. \begin{aligned} \frac{d}{dt} \mathbf{V}_i + (\mathbf{V}_i \cdot \nabla_h) \mathbf{V}_i + \nabla_h P_i - \nu \Delta_h \mathbf{V}_i - \mathbf{f}(\mathbf{x}_i, t) &= 0 \\ \Delta_h P_i - \sum_m \nabla_h V_{m,i} \cdot D_{m,h} \mathbf{V}_i - C_{d,i} \nabla_h \cdot \mathbf{V}_i - \nabla_h \cdot \mathbf{f}(\mathbf{x}_i, t) &= 0 \end{aligned} \right\} \quad \mathbf{x} \in \Omega$$

$$\left. \begin{aligned} B(\mathbf{V}_i, P_i) &= 0 \\ \nabla_h \cdot \mathbf{V}_i &= 0 \end{aligned} \right\} \quad \mathbf{x}_i \in \partial\Omega_h$$

$$\mathbf{V}(\mathbf{x}_i, 0) = \mathbf{U}_0(\mathbf{x}_i) \quad \text{at } t = 0$$

Extra *numerical boundary conditions* are added for the second- or fourth-order accurate spatial discretisations.

Cgins: time-stepping with Adams predictor-corrector methods

The incompressible N-S equations can be written as

$$\mathbf{u}_t = \mathbf{f}(\mathbf{u}, p)$$

where the pressure p is considered to be function of \mathbf{u} . A method-of-lines approach may be used to integrate the equations.

The **second-order accurate Adams predictor-corrector** time stepping method for the INS equations can be chosen with the “adams PC” option. It is defined by

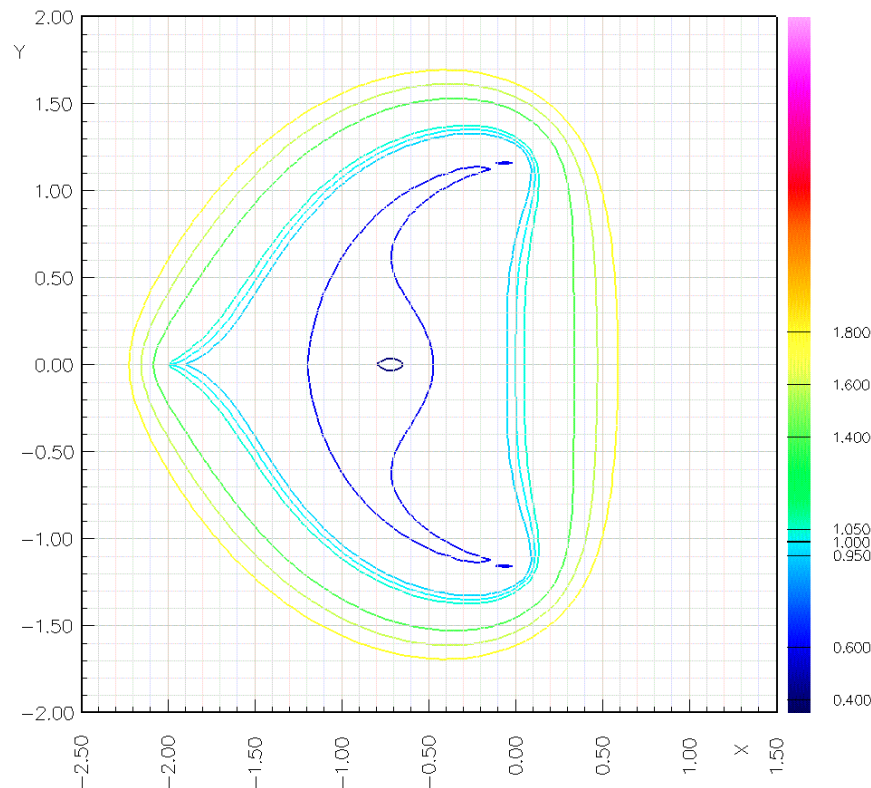
$$\begin{aligned}\frac{\mathbf{u}^p - \mathbf{u}^n}{\Delta t} &= \frac{3}{2}\mathbf{f}^n - \frac{1}{2}\mathbf{f}^{n-1} \\ \frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} &= \frac{1}{2}\mathbf{f}^p + \frac{1}{2}\mathbf{f}^n\end{aligned}$$

where one correction step has been used (one may optionally correct more than one time).

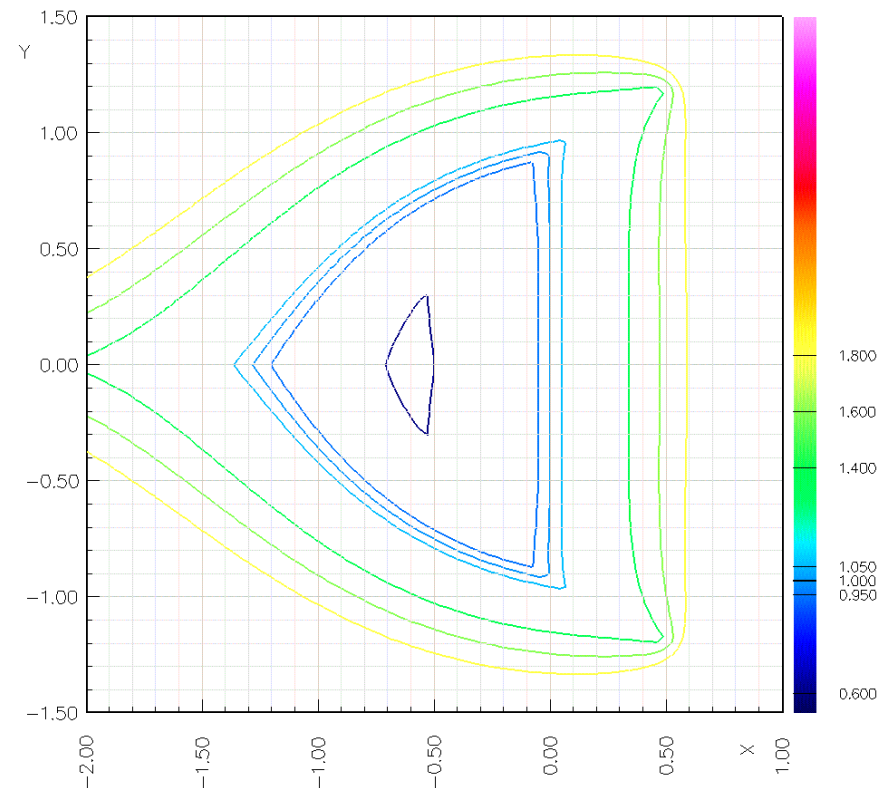
For fourth-order accuracy in space, one may choose a time-stepping method that is order 2,3 or 4.

Cgins: stability regions for some Adams PC methods

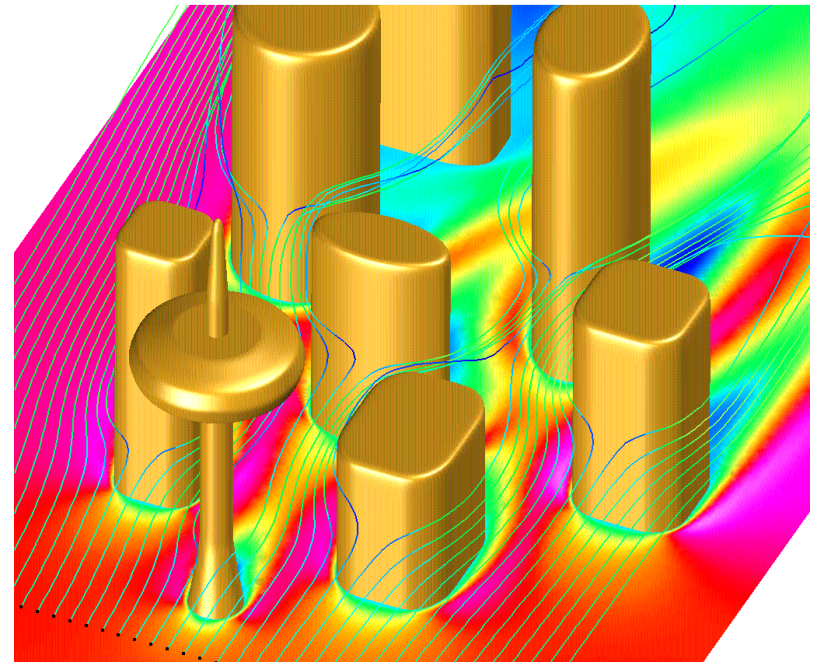
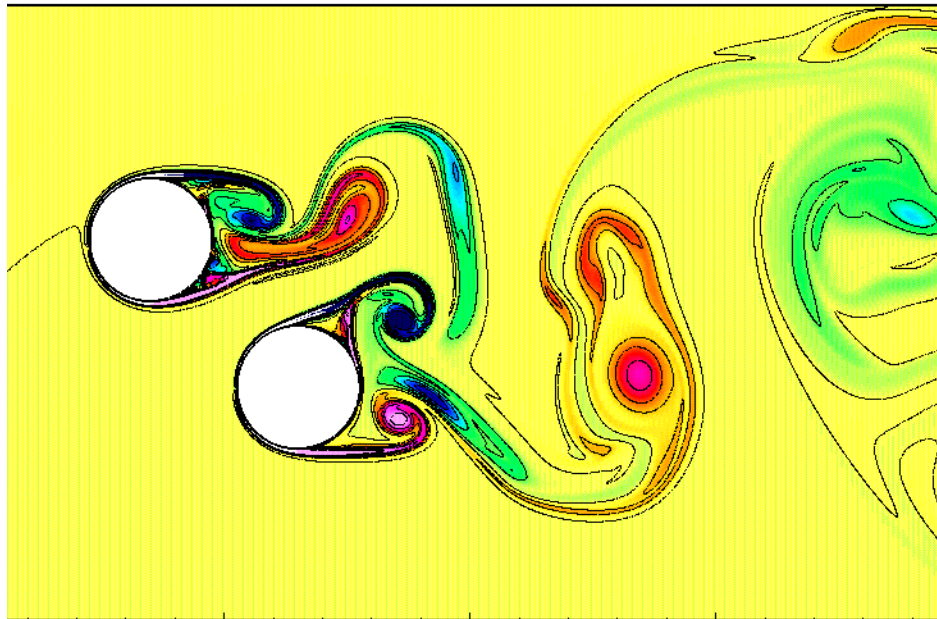
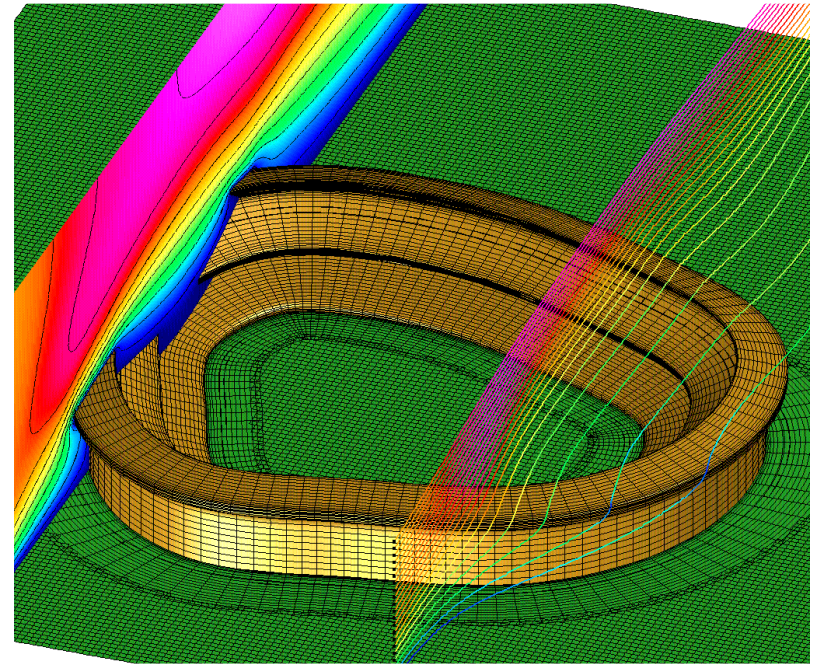
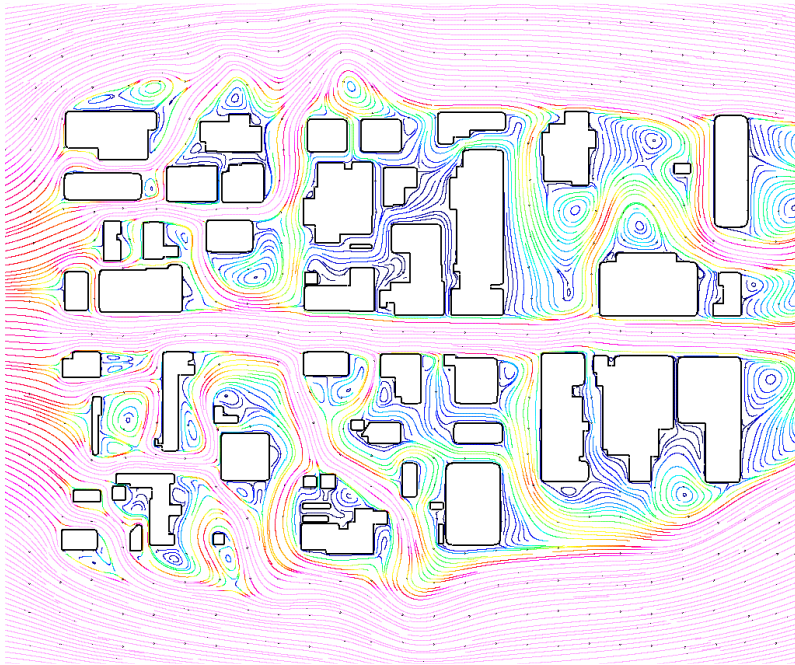
Stability Region: 2-2 Predictor Corrector



Stability Region: 4-4 Predictor Corrector



Stability regions for the predictor corrector methods. Left: second-order method, PECE mode. Right: fourth-order method, PECE mode. The region of stability is inside the contour level value of 1.



Incompressible flow computations with cgins.

Cgcns: Reactive Euler Equations

Governing equations (2-D):

$$\mathbf{u}_t + \mathbf{f}(\mathbf{u})_x + \mathbf{g}(\mathbf{u})_y = \mathbf{h}(\mathbf{u})$$

where

$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \\ \rho \lambda \end{bmatrix} \quad \mathbf{f} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(\rho E + p) \\ \rho u \lambda \end{bmatrix} \quad \mathbf{g} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(\rho E + p) \\ \rho v \lambda \end{bmatrix} \quad \mathbf{h} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \rho R \end{bmatrix}$$

Variables:

$$\begin{array}{ll} \rho = \text{density} & (u, v) = \text{velocity} \\ p = \text{pressure} & E = \text{total energy} \\ \lambda = n \text{ mass fractions} & R = n \text{ reaction rates} \end{array}$$

$$E = e + \frac{1}{2}(u^2 + v^2)$$

where

$$e = e(\rho, p, \lambda) = \begin{cases} \text{internal energy per unit mass} \\ \text{(as specified by an equation of state)} \end{cases}$$

Cgcns: Reactive Euler Equations

To reactive Euler equations are written in conservation form in terms of the unit square coordinates, $\mathbf{r} = (r_1, r_2)$

$$\mathbf{u}_t + \frac{1}{J} \left(\frac{\partial \mathbf{F}}{\partial r_1} + \frac{\partial \mathbf{G}}{\partial r_2} \right) = \mathbf{h},$$

where

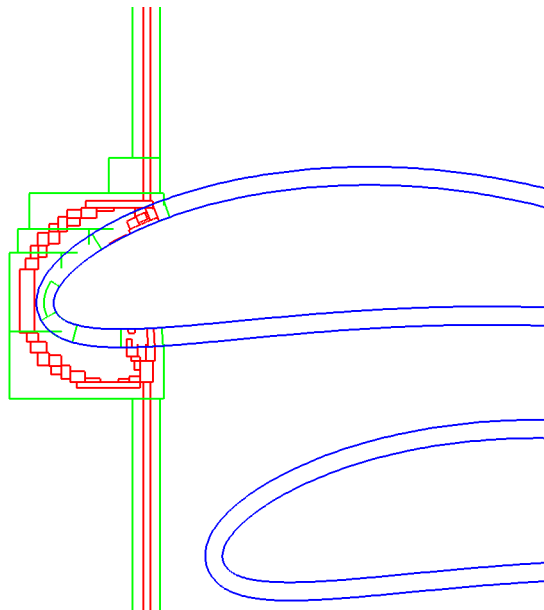
$$\mathbf{F} = J \nabla r_1 \cdot (\mathbf{f}, \mathbf{g}),$$

$$\mathbf{G} = J \nabla r_2 \cdot (\mathbf{f}, \mathbf{g}),$$

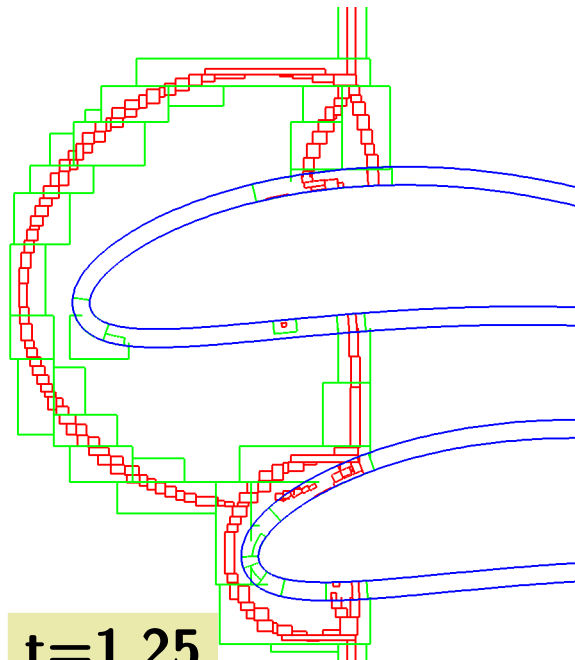
$$J = \det \left[\frac{\partial \mathbf{x}}{\partial \mathbf{r}} \right].$$

This new system is solved on the unit square and discretised with a second-order Godunov method. The reaction terms are treated with a sub-cfl time-step Runge-Kutta method with error control. This discretization was developed by Don Schwedeman (RPI) [10].

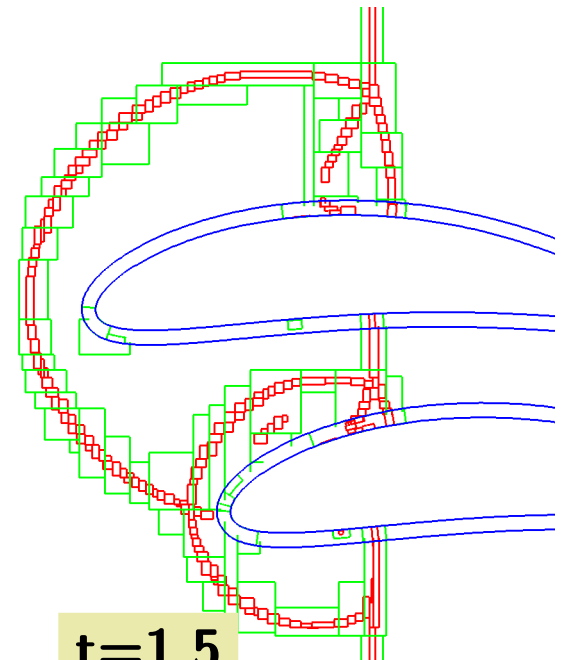
Adaptive overlapping grids



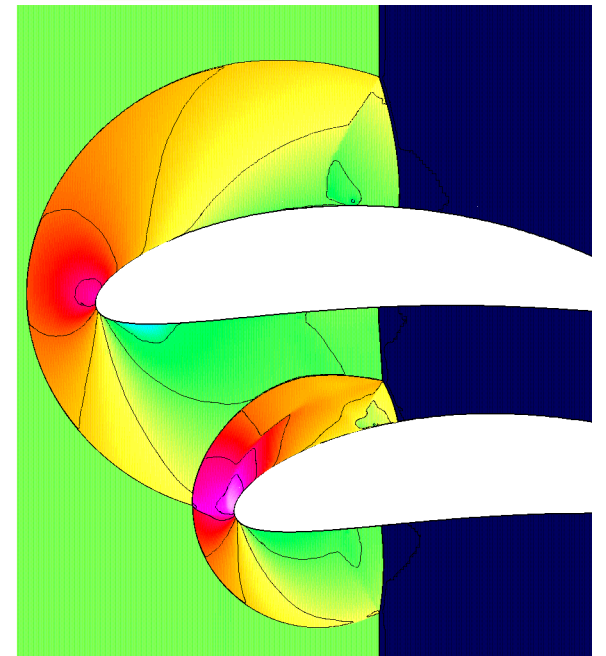
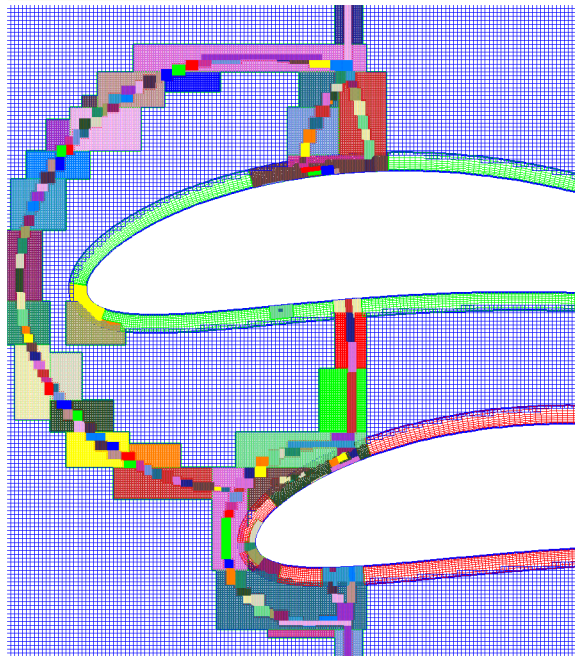
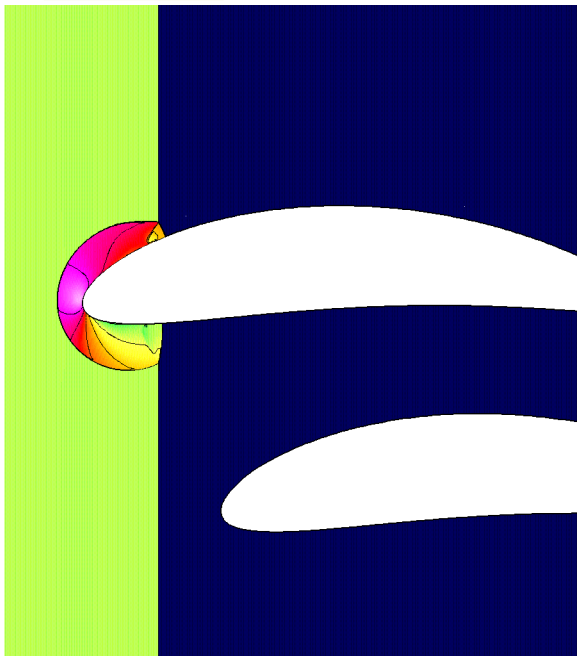
$t=0.75$

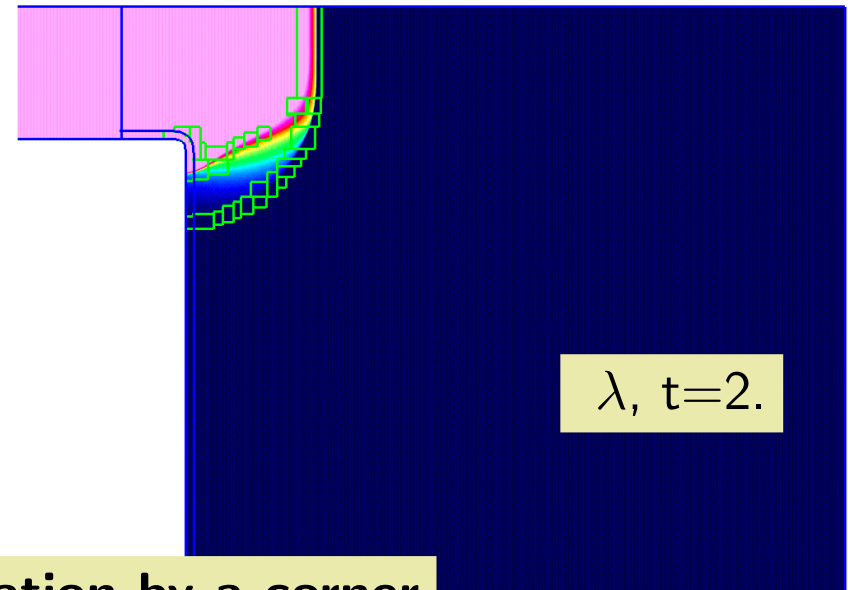
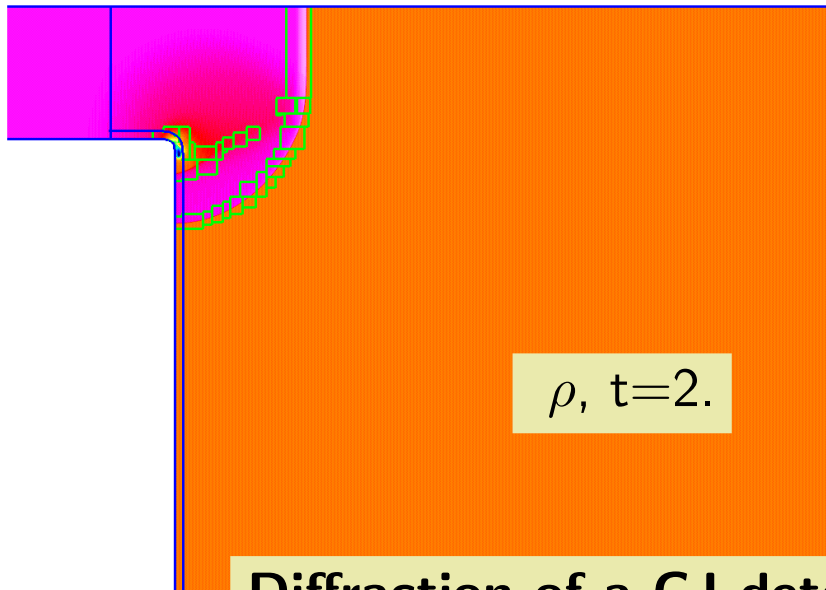


$t=1.25$



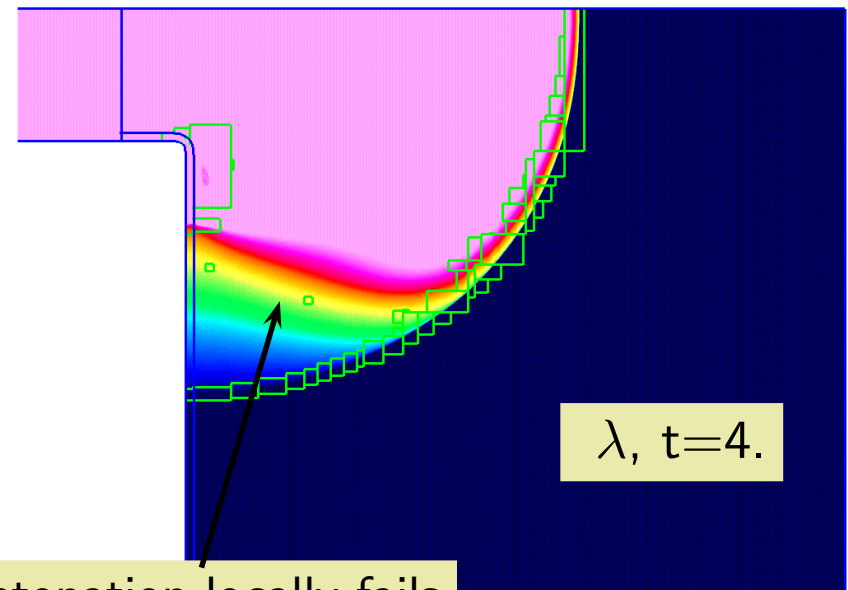
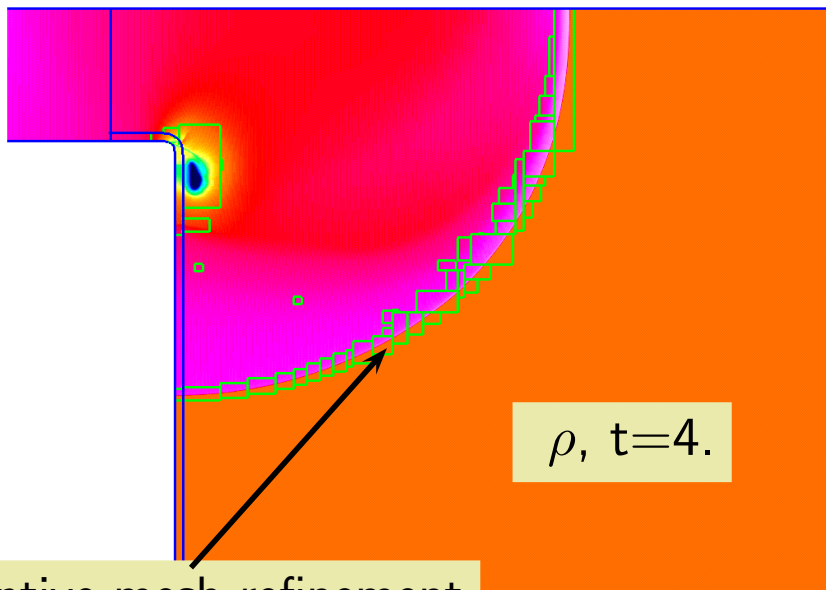
$t=1.5$





Diffraction of a CJ detonation by a corner

LX-17, ignition-and-growth model, JWL equation of state.

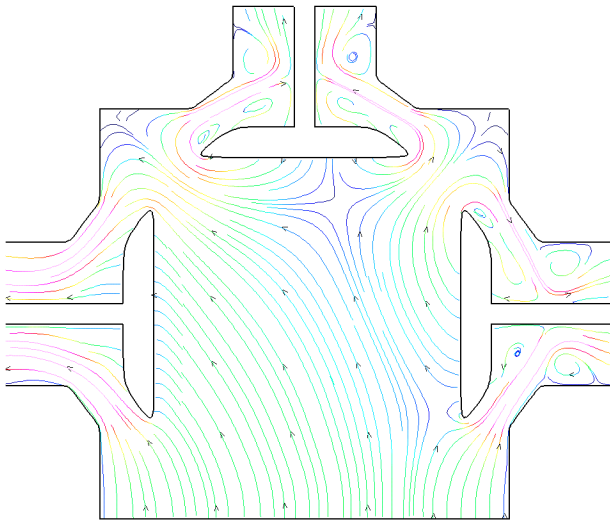


adaptive mesh refinement

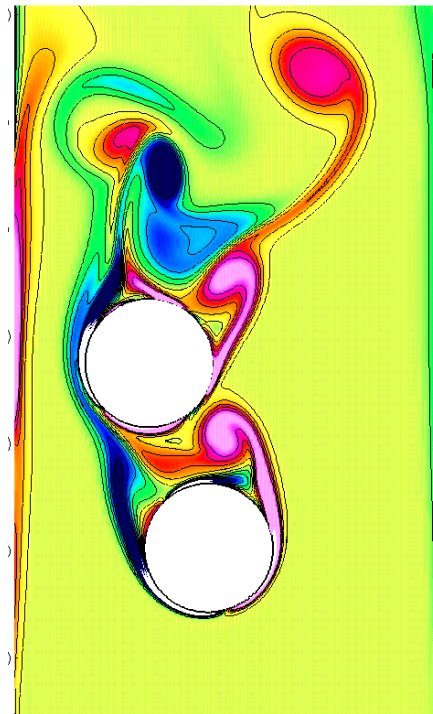
detonation locally fails

Current work: moving geometry

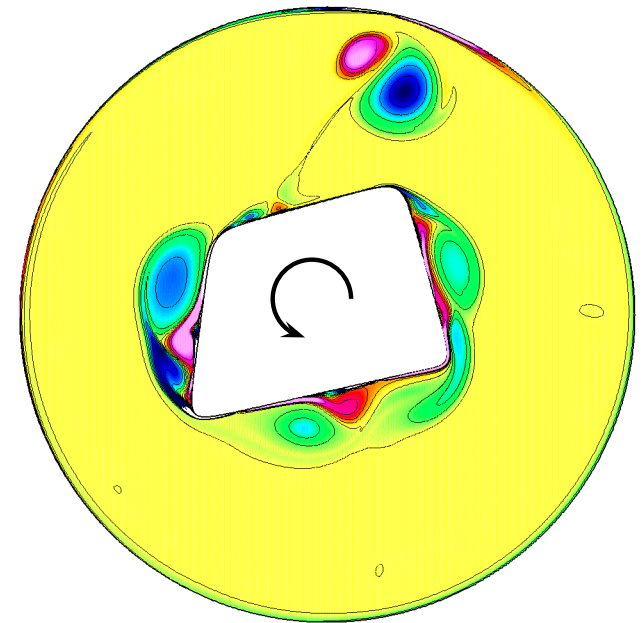
- ◇ The governing equations are written in the moving coordinate system
- ◇ Support for (1) specified motion, (2) rigid-body motion with forces and torques determined from the flow.
- ◇ The grids are moved at every time step and the interpolation points are recomputed.



Moving valves (INS)

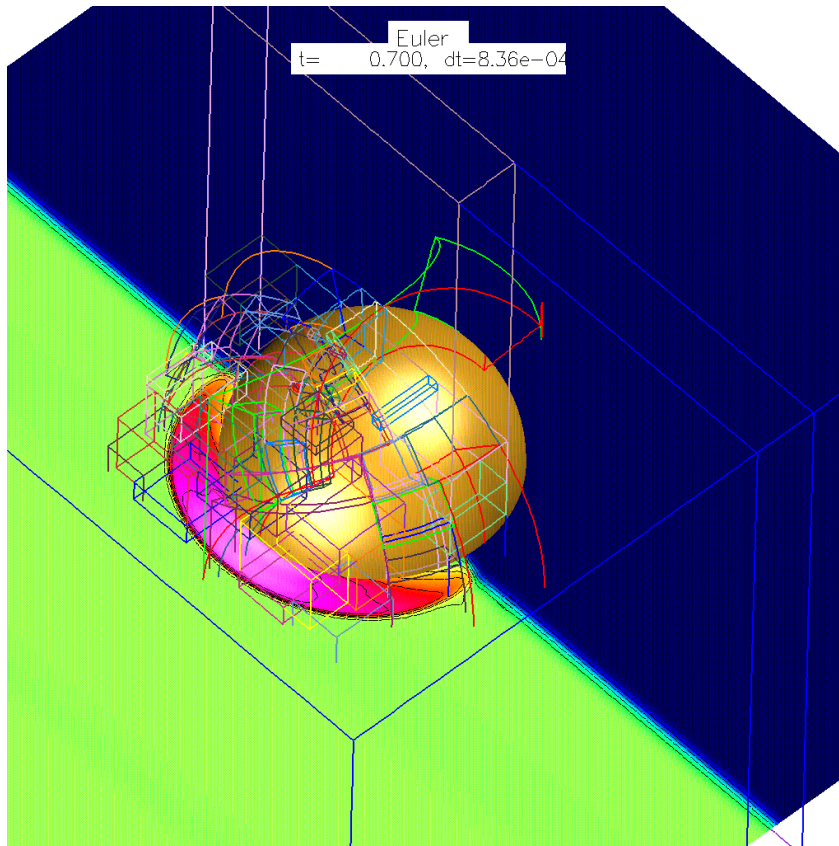


Falling cylinders (INS)

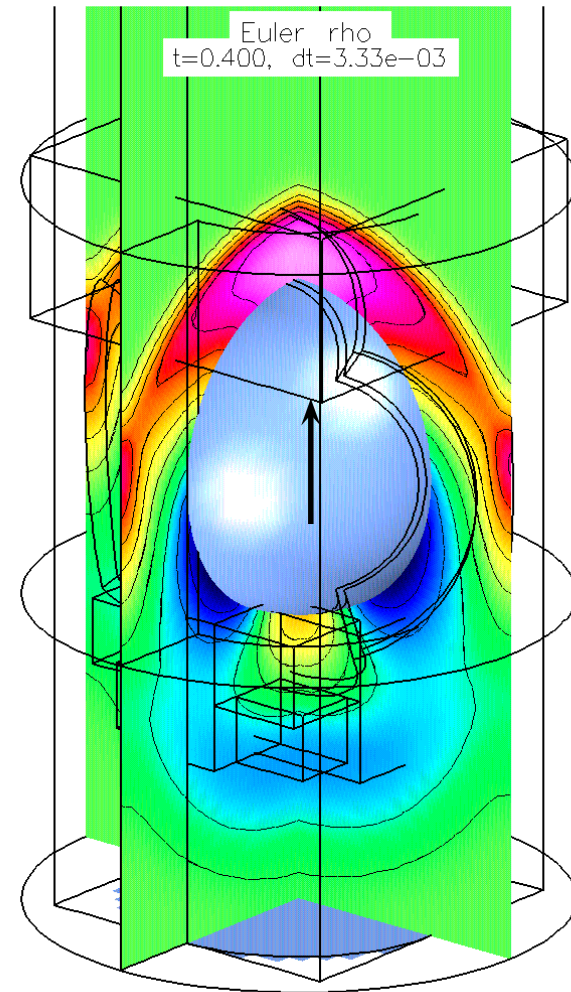


Rotating body (INS)

Current work: three-dimensions



Shock hitting a sphere (Euler)



Sphere moving in a tube with AMR (Euler)

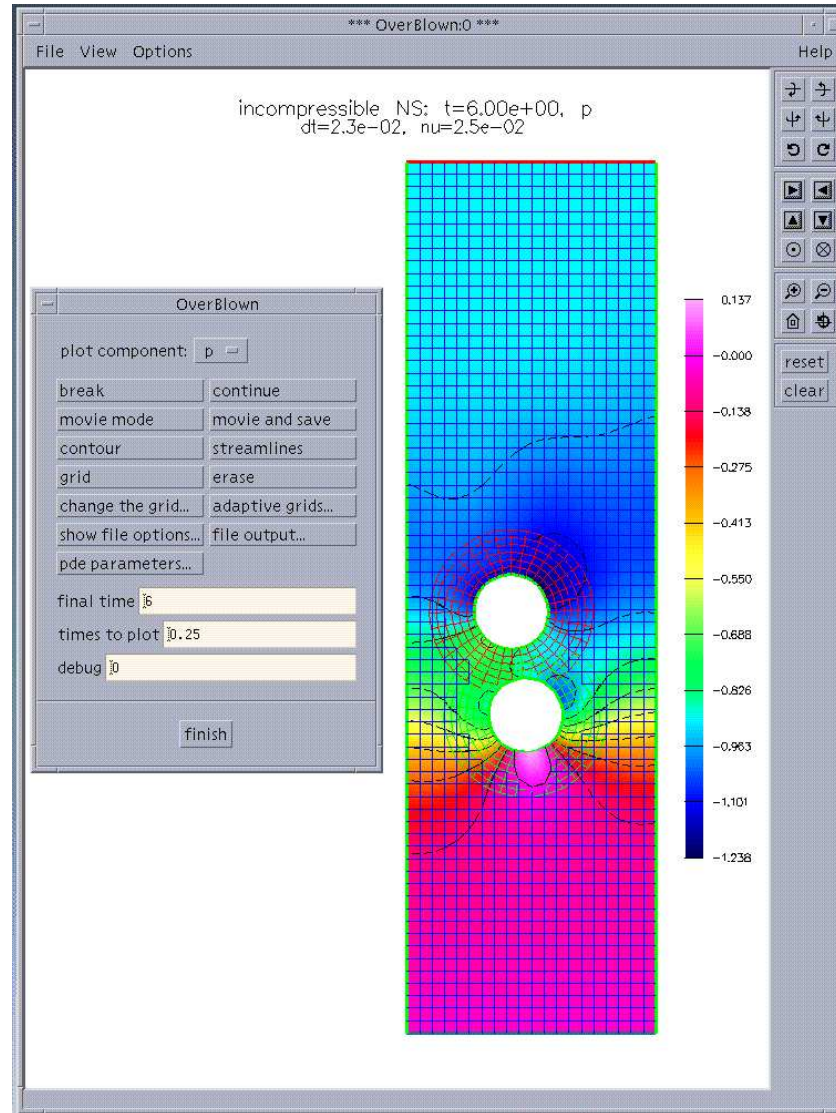
Current work: parallel versions of the CG solvers

- ◇ Grids can be distributed across one or more processors.
- ◇ Distributed parallel arrays using P++ (K. Brislawn, B. Miller, D. Quinlan)
- ◇ P++ uses Multiblock PARTI (A. Sussman, G. Agrawal, J. Saltz) for block structured communication with MPI (ghost boundary updates, copies between different distributed arrays)
- ◇ A special parallel overlapping grid interpolation routine has been written.

NP	sec/step	ratio
1	8.4	1.
2	4.3	2.

Table 4: Shock hitting a cylinder (no AMR), 1.1 million grid-points, Dell workstation 2.2GHz Xeon.

Running the CG solvers...



Snapshot of cgin showing the run time dialog menu. The figure shows two falling bodies in an incompressible flow (using the command file twoDrop.cmd).

A cgins command file (script)...

```
*
* Two dropping cylinders
*
* Name of the grid:
  twoDrop.hdf
* Equations to solve:
  incompressible Navier Stokes
exit
*
show file options
  open
    twoDrop.show
  frequency to flush
    4
  exit
*
turn off twilight zone
project initial conditions
*
turn on moving grids
detect collisions
specify grids to move
  rigid body
```

```
density
  .5
moments of inertia
  1.
done
drop
done
rigid body
density
  .5
moments of inertia
  1.
done
drop2
done
done
* use implicit time stepping
implicit
choose grids for implicit
  all=implicit
  channel=explicit
done
*
```

```
pde parameters
  nu
    .025
  * turn on gravity
  gravity
    0. -1. 0.
done
*
boundary conditions
  all=noSlipWall
  channel(1,1)=inflowWithVelocity
  channel(0,1)=outflow
done
*
initial conditions
  uniform flow
  p=1.
exit
*
final time (tf=)
  8. 6.
times to plot (tp=)
  .1
```


Demo: cgins, cgns

- INS: flow past a cylinder (or two).
- INS: two falling cylinders.
- Euler: shock hitting a cylinder (using AMR).

References

- [1] G. Chesshire and W.D. Henshaw. Composite overlapping meshes for the solution of partial differential equations. *J. Comp. Phys.*, 90(1):1–64, 1990.
- [2] G. Chesshire and W.D. Henshaw. A scheme for conservative interpolation on overlapping grids. *SIAM J. Sci. Comput.*, 15(4):819–845, July 1994.
- [3] B. Gustafsson, H.-O. Kreiss, and A. Sundström. Stability theory of difference approximations for mixed initial boundary value problems. II. *Mathematics of Computation*, 26(119):649–686, 1972.
- [4] Bertil Gustafsson, Heinz-Otto Kreiss, and Joseph Oliger. *Time Dependent Methods and Difference Methods*. John Wiley and Sons Inc., 1995.
- [5] W.D. Henshaw. *Part II: Composite Overlapping Grid Techniques*. PhD thesis, Dept. of Applied Mathematics, California Institute of Technology, 1985.
- [6] W.D. Henshaw. A fourth-order accurate method for the incompressible Navier-Stokes equations on overlapping grids. *J. Comp. Phys.*, 113(1):13–25, July 1994.
- [7] W.D. Henshaw and G. Chesshire. Multigrid on composite meshes. *SIAM J. Sci. Stat. Comput.*, 8(6):914–923, 1987.
- [8] W.D. Henshaw, H.-O. Kreiss, and L.G.M. Reyna. A fourth-order accurate difference approximation for the incompressible Navier-Stokes equations. *Comput. Fluids*, 23(4):575–593, 1994.

- [9] William. D. Henshaw. On multigrid for overlapping grids. Research Report UCRL-JRNL-201940, Lawrence Livermore National Laboratory, 2003. Submitted for publication.
- [10] William. D. Henshaw and Donald W. Schwendeman. An adaptive numerical scheme for high-speed reactive flow on overlapping grids. *J. Comp. Phys.*, 191:420–447, 2003.
- [11] M. Hinatsu and J.H. Ferziger. Numerical computation of unsteady incompressible flow in complex geometry using a composite multigrid technique. *International Journal for Numerical Methods in Fluids*, 13:971–997, 1991.
- [12] H.-O. Kreiss and J. Lorenz. *Initial-Boundary Value Problems and the Navier-Stokes Equations*. Academic Press, 1989.
- [13] G. Starius. Composite mesh difference methods for elliptic and boundary value problems. *Numer. Math.*, 28:243–258, 1977.
- [14] G. Starius. On composite mesh difference methods for hyperbolic differential equations. *Numer. Math.*, 35:241–255, 1980.
- [15] J.C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Wadsworth and Brooks/Cole, 1989.
- [16] K. Stüben and U. Trottenberg. Multigrid methods: Fundamental algorithms, model problem analysis and applications. In W. Hackbusch and U. Trottenberg, editors, *Multigrid Methods*, pages 1–176. Springer-Verlag, 1982.
- [17] U. Trottenberg, C.W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, London,

2001.

- [18] J. Y. Tu and L. Fuchs. Overlapping grids and multigrid methods for the three-dimensional unsteady flow calculations in IC engines. *International Journal for Numerical Methods in Fluids*, 15(6):693–714, 1992.
- [19] J. Y. Tu and L. Fuchs. Calculation of flows using three-dimensional overlapping grids and multigrid methods. *International Journal for Numerical Methods in Engineering*, 38:259–282, 1995.
- [20] Y. Zang and R.L. Street. A composite multigrid method for calculating unsteady incompressible flows in geometrically complex domains. *International Journal for Numerical Methods in Fluids*, 20:341–361, 1995.